15-451/651 Algorithms, Spring 2018

Homework #5 **Due: April 3, 2018**

Same instructions as previous written HWs.

(25 pts) 1. **Safety First!**

It is 1944, and you are a physicist working on the Manhattan project, attempting to build an atomic bomb. In the process, you will be enriching a lot of uranium. The thing about this enriched uranium is, if too much of it gets close together, it will explode, making you very unhappy. Naturally, you will take precautions to make sure you don't get blown up, though ideally you'd like to do it as inexpensively as possible...

(a) The first approach is to take your storage facilities as fixed, and place the uranium in locations that are sufficiently spaced out. We model the storage facility as an undirected graph $G = (V, E)$, where V is the set of locations at which we can store uranium. The edges have length $\ell(e)$, which induce a distance metric on the graph, where $d(u, v)$ is the cost of the minimum length path from u to v. You are given a parameter D. Putting uranium at two locations u and v such that $d(u, v) < D$ will cause an explosion. You'd like to place as many units of uranium as possible down, subject to the constraint of not being blown up. Derive a polynomial time algorithm to maximize the number of units of uranium you can place (the output is the set of locations where you'd put it), or show that this problem is NP-hard.

(b) Another way to control the nuclear chain reaction is with control rods that stop the cascading neutrons. So you can imagine another undirected graph $G = (V, E)$ where you plan to place uranium at each and every location in V. This is safe if and only if you place control rods down at a set of locations $C \subseteq V$ such that each $v \in V$ is either in C, or has a neighbor in C. Your goal is the make the site safe while minimizing the size of C. Derive a polynomial time algorithm to minimize $|C|$, or show that this problem is NP-hard.

(c) A third way to deal with the uranium problem is to reinforce the walls with material that stops neutrons. Here you have an undirected graph $G = (V, E)$ with non-negative edge weights $w(e)$ and with a set $Y \subseteq V$ of locations where you plan to put the uranium. Each edge e represents a barrier to neutrons that can be reinforced any amount $\ell(e) \in \mathbb{R}_{\geq 0}$ at a cost of $w(e)\ell(e)$. You must reinforce the edges of the graph such that for any two vertices $u, v \in Y$, the shortest path from u to v under these lengths $\ell(e)$ (i.e., where you interpret edge e as having length $\ell(e)$) is at least one. Your goal is to find values of $\ell(e)$ to minimize the total cost you pay, subject to not being blown up. Derive a polynomial time algorithm to minimize the cost, or show that this problem is NP-hard.

(25 pts) 2. **What are Widgets, Anyways?**

You are in charge of the supply chain for Widgets-r-Us, where you have to find ways to transport materials from their storage warehouses to the factories, subject to capacity constraints on the transportation network. You model it as a directed graph $G = (V, E)$, where each directed edge $e \in E$ represents a one-way road. (Roads capable of carrying traffic in both directions can be thought of as two one-way roads.)

There are k "warehouse" vertices, and k kinds of materials, where the i^{th} material is originally located at the warehouse vertex $s_i \in V$, and the total amount of material i is D_i. Each road (directed edge) e has a capacity u_e, such that the total amount of material (of all kinds) sent over this road must be at most u_e. You may assume that all quantities given as input in this problem are non-negative integers.

You also have ℓ "factory" vertices. There are ℓ factories producing widgets (the j^{th} factory is located at the j^{th} factory vertex $f_j \in V$). The factory f_j has a request vector $\mathbf{r}_j = (r_{1j}, r_{2j}, \ldots, r_{kj})$, such that to produce one unit of widget, it requires r_{ij} amounts of material i for every $i \in \{1, \ldots, k\}$. [1]

(a) Write an LP to figure out how to transport the material from their warehouses to the factories (respecting the road capacity constraints), to maximize the total amount of widgets produced, subject to these constraints. (It is OK if you produce fractional amounts of widgets.)

(b) You find out that widgets produced at different factories sell for different amounts of money: the price per unit of widget produced at factory j is p_j. You want to maximize your revenue. Change the LP from the previous part to handle this.

(c) You are informed that two of the roads $e_1 = (u, v)$ and $e_2 = (v, u)$ are special: they represent the two directions of traffic over a bridge. For structural reasons, you have the balance requirement that the absolute value of the difference between the amount of material send in the two directions can be at most $\delta \in \mathbb{Z}_{\geq 0}$. How can you add such a constraint to your LP.

(25 pts) 3. **(It's in the Bag.)** Consider the knapsack problem we solved using dynamic programming in time $O(nS)$. (See Lecture #8 for definitions; assume each item has size at most S.) This is not good if S is very large.

(a) Give a dynamic-programming algorithm with running time $O(nV^*)$, where V^* is the value of the *optimal solution*. So, this would be a better algorithm if the sizes are much larger than the values.

Note: your algorithm should work even if V^* is not known in advance. You may want to first assume you are given V^* up front and then afterwards figure out how to remove that requirement.

[1] Nodes like s_i and f_j can have both incoming and outgoing arcs, and the route taking material i from the warehouse s_i to some factory f_j is allowed to pass through some other warehouse vertex $s_{i'}$ or some other factory vertex $f_{j'}$. You may assume that all the $k + \ell$ warehouse and factory vertices are distinct.

(b) Now given an instance I of knapsack and some real $k \geq 1$, define new values $v'_i := k \cdot \lfloor \frac{v_i}{k} \rfloor$. This gives a new instance I'. Since item sizes and S remain the same, clearly the feasible solutions to I and I' are the same.

For any feasible solution, let its value in I be V, and its value in I' be V'. Show that $V \geq V' \geq V - nk$.

(c) Use part (a) to show that I' can be solved in at most $O(\frac{n^2 v_{\max}}{k})$ time.

(d) Given any knapsack instance I and a value $\epsilon \in (0, 1)$, show that setting $k := \frac{\varepsilon\, v_{\max}}{n}$ gives an algorithm that returns a feasible solution to I, has value least $(1 - \varepsilon)$ times the optimal value of I, and runs in time $O(\frac{n^3}{\varepsilon})$.

In other words, if you wanted to find a solution whose value is within 99% *of the optimum value, use this algorithm with* $\varepsilon = 0.01$.

(25 pts) 4. **Travelling Salesperson Problem**

In this problem you will implement an algorithm that can solve the Travelling Salesperson Problem (TSP). We discussed this problem in Lecture #9 and an algorithm based on subset dynamic programming. The input to your program will be a connected graph G with n vertices and m edges. Each edge $\{u, v\}$ has a non-negative length in each direction, and $\text{len}(u, v)$ may be different from $\text{len}(v, u)$. Your program will compute the length of the shortest TSP tour for the given graph, and also output the tour. The time limit is 10 seconds. The filename is `tsp.c`, or analogues thereof. Note you do not need to provide a written explanation of your algorithm.

Input: The first line consists of two positive integers n and m. $n \leq 20$ and $n - 1 \leq m \leq n(n-1)/2$. The following m lines each contain four integers that represent an edge. These numbers are i j $d_{i,j}$ $d_{j,i}$. This means that there is an edge $\{i, j\}$ in the graph and the directed length from i to j is $d_{i,j}$ and the directed length from j to i is $d_{j,i}$. Each edge will occur at most once in the list in either direction. There are no self loops, and the graph is connected. The vertex numbers are in $[0, n-1]$, and the lengths of the edges are in $[0, 10^6]$.

Output: The first line of output contains the cost of the shortest travelling salesperson tour for the given graph. The second line contains list of the cities in the order visited, starting at city 0 and ending at city 0. Each pair of neighboring cities on this list must be an edge in G. Any tour which is of minimum cost is acceptable.

Samples: For example, if the input is:

```
5 5
0 1 1 1
1 2 1 1
2 3 1 2
3 4 1 1
4 0 1 1
```

Then the output (which is unique in this case) will be:

```
optimal tour cost = 5
tour: 0 1 2 3 4 0
```

For example, if the input is:

```
6 6
0 1 1 1
1 2 1 1
2 3 1 1
3 1 1 2
2 4 1 1
3 5 1 1
```

Then the output might be:

```
optimal tour cost = 9
tour: 0 1 2 4 2 3 5 3 1 0
```

(1 bonus) B4. **Magic Boards. (1 bonus pt)**

Arthur is given an $n \times n$ square grid, with some of the cells already filled in with Xs and Os, and some empty cells. He is also given a target positive integer z. To win, Arthur must place at least z many new X's on the grid (at most one per empty cell), subject to the following constraints.

- The number of X's in the i^{th} row must equal the number of X's in the i^{th} column, for each i.
- Arthur is also given a real value $0 < \alpha < 1$, and must ensure that no row or column contains more than an α fraction of all the X's on the board.

Give a polynomial-time algorithm that will output a winning solution if it is possible for him to win, and else output `Impossible` if there is no way for him to win.

(1 bonus) B5. **Randomness-Efficient Hashing. (1 bonus pt)**

Suppose we have m distinct integers $x_1, \ldots, x_m$ from the universe $U = \{0, 1, 2, 3, ..., n-1\}$. We would like to choose a random hash function h from a family H so that for all $i \neq j$, $h(x_i) \neq h(x_j)$, that is, the set $\{x_1, \ldots, x_m\}$ is perfectly hashed under h. We saw that if H is universal and has range size $M = \Theta(m^2)$, then a random h in H has this property with constant probability. However, for each of the hash functions we saw, if $x_1, \ldots, x_m$ are not fixed in advance, then specifying a random h in H requires at least $\log n$ bits. We would like to use fewer random bits when m is much smaller than n.

Suppose we choose a random prime p among the first $10m^2 \cdot \log_2 n$ primes. Consider the map $h : U \to \{0, 1, 2, ..., 20p^2 - 1\}$ given by $h(y) = g(y \bmod p)$, where g is drawn from a universal hash function family of functions with domain $\{0, 1, 2, \ldots, p-1\}$ and range $\{0, 1, 2, \ldots, 10m^2 - 1\}$. You can assume the existence of such universal hash families. Argue that h has the above perfect hashing property with probability at least $9/10$. How many bits are needed to specify h?