ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Homework I, Advanced Algorithms 2018

Due on Thursday March 29 at 17:00 (send an email to ola.svensson@epfl.ch). Solutions to many homework problems, including problems on this set, are available on the Internet, either in exactly the same formulation or with some minor perturbation. It is *not acceptable* to copy such solutions. It is hard to make strict rules on what information from the Internet you may use and hence whenever in doubt contact Ola Svensson. You are, however, allowed to discuss problems in groups of up to three students; it is sufficient to hand in one solution per group.

Hint: Recall that if E is any finite subset of a vector space $\mathbb{V}$, then we can define a matroid M on E by taking the independent sets of M to be the linearly independent subsets of E. This is called a linear matroid. Note that the vector space $\mathbb{V}$ can be over any field, e.g. the real numbers $\mathbb{R}$, the finite field $GF(2)$, the complex numbers $\mathbb{C}$, etc.

In your answers, you are allowed to use any algorithm that was stated in the course, such as the polynomial-time algorithm for finding a max-weight independent set in the intersection of two matroids[1].

1 *(15 pts)* **The class partition problem.** In this exercise we will consider a problem that we call the class partition problem. We are given a set C of kids and a non-empty set P consisting of pairs of kids that have bad influence on each other. The goal is to partition the kids in C into $k \geq 2$ classes $T_1, T_2, \ldots, T_k$ so as to maximize the number of problematic pairs that are not placed in the same class, i.e., we wish to maximize

$$\text{val}(T_1, \ldots T_k) := |\{(a,b) \in P : a \in T_i \text{ and } b \in T_j \text{ for } i \neq j\}|\,.$$

A hint for both subproblems is to analyze a randomized algorithm.

1a *(10 pts)* Show that there always exists a partitioning $T_1, \ldots, T_k$ of the kids such that

$$\text{val}(T_1, \ldots T_k) \geq \left(1 - \frac{1}{k}\right)|P|\,.$$

Solution: The algorithm we use is very simple: put each child $c \in C$ into one of the k classes independently and uniformly at random. We will now analyse the expected value of the returned solution. Let Ω be the set of all possible partitions of the kids into classes and $X : \Omega \to \mathbb{N}$ a random variable representing the number of problematic pairs of kids that are separated:

$$X(T_1, T_2, \ldots, T_k) = |\{(a,b) \in P : a \in T_i \text{ and } b \in t_j \text{ for } i \neq j\}|$$

[1] To be precise, this holds assuming that we have efficient independence oracles for the matroids, which is true for all matroids we encountered in class.

If we can show that $\mathbb{E}[X] = |P|(1-1/k)$, then this implies that there exists some $\omega \in \Omega$ such that $X(\omega) \geq |P|(1-1/k)$ because by definition, $\mathbb{E}[X] = \sum_{\omega\in\Omega} \Pr(\omega)X(\omega)$. To compute the expectation of X without first having to compute the probability distribution, we introduce for each $(a,b) \in P$ a binary random variable X_{ab} which is set to 1 if kids a and b are in different classes and 0 else. This trick allows us to write $X = \sum X_{ab}$ and thus, by linearity of expectation:

$$\mathbb{E}[X] = \sum_{(a,b)\in P} \mathbb{E}[X_{ab}] = \sum_{(a,b)\in P} \Pr(\text{"}a \text{ and } b \text{ are in different classes"})$$

Computing this probability is quite straightforward:

$$\begin{aligned}\Pr(\text{"}a \text{ and } b \text{ are in different classes"}) &= 1 - \Pr(\text{"}a \text{ and } b \text{ are in the same classes"}) \\ &= 1 - \sum_{i=1}^{k} \Pr(\text{"}a \text{ and } b \text{ are both in class } i) \\ &= 1 - 1/k\end{aligned}$$

So $\mathbb{E}[X] = |P|(1-1/k)$ and we're done.

1b *(5 pts)* Show that there always exists a partitioning $T_1, \ldots, T_k$ of the kids such that

$$\text{val}(T_1, \ldots T_k) > \left(1 - \frac{1}{k}\right)|P|\,.$$

(Notice the strict inequality.)

Solution: To prove that there exists some $\omega \in \Omega$ such that $X(\omega) > \mathbb{E}[X]$, we will need a finer analysis. If we can show that the variance of X is non-zero then we're good:

Lemma 1. *If* $\text{Var}[X] > 0$*, then there exists* $\omega \in \Omega$ *such that* $X(\omega) > \mathbb{E}[X]$

Proof. Suppose toward contradiction that for all $\omega \in \Omega$, $X(\omega) \leq \mathbb{E}[X]$. Using the definition of expectation we have:

$$\mathbb{E}[X] = \sum_{\omega\in\Omega} \Pr(\omega)X(\omega) \leq \sum_{\omega\in\Omega} \Pr(\omega)\mathbb{E}[X] = \mathbb{E}[X]$$

So the inequality is actually an equality and $X(\omega) = \mathbb{E}[X]$ for all $\omega \in \Omega$. Therefore, the variance is zero: a contradiction. □

Instead of computing the variance directly using the indicator random variables again, simply notice that $\Pr(X = 0) = k^{1-n}$ which is strictly larger than zero. Thus, there exists $\omega \in \Omega$ such that $X(\omega) \neq \mathbb{E}[X]$ (provided that $k \geq 2$). Therefore, we conclude that $\text{Var}[X] > 0$.

Note that this kind of non-constructive proofs are part of the *probabilistic method*, which has many other cool applications.

2 *(15 pts)* **Primal-dual algorithm for the weighted Vertex Cover problem.** Let $G = (V, E)$ be a Vertex Cover instance with vertex weights $w : V \to \mathbb{R}_+$. Recall the LP relaxation and its dual:

(Primal) LP Relaxation	(Dual)
minimize $\sum_{v \in V} w(v) x_v$	**maximize** $\sum_{e \in E} y_e$
subject to $x_u + x_v \geq 1$ for $\{u, v\} \in E$	**subject to** $\sum_{u:\{u,v\} \in E} y_{\{u,v\}} \leq w(v)$ for $v \in V$
$x_v \geq 0$ for $v \in V$	$y_e \geq 0$ for $e \in E$

The Vertex Cover problem is NP-hard and therefore we do not expect an efficient (polynomial-time) algorithm that finds exact solutions. In this problem, we will analyze and implement a simple and very fast primal-dual algorithm that achieves the best-known guarantees.

The primal-dual algorithm works as follows. It will maintain a feasible dual solution y that initially is set to $y_e = 0$ for every $e \in E$. It will then iteratively improve the dual solution until the set $C = \{v \in V : \sum_{u:\{u,v\} \in E} y_{\{u,v\}} = w(v)\}$ forms a vertex cover. Note that C consists of those vertices whose constraints in the dual are tight. The formal description of the algorithm is as follows:

1. Initialize the dual solution y to be $y_e = 0$ for every $e \in E$.
2. While $C = \{v \in V : \sum_{u:\{u,v\} \in E} y_{\{u,v\}} = w(v)\}$ is not a vertex cover:
   - Select an edge $\{u, v\} \in E$ that is not covered by C, i.e., $C \cap \{u, v\} = \emptyset$.
   - Increase $y_{\{u,v\}}$ until one of the dual constraints (corresponding to u or v) becomes tight.
3. Return $C = \{v \in V : \sum_{u:\{u,v\} \in E} y_{\{u,v\}} = w(v)\}$.

Show that the primal-dual algorithm has an approximation guarantee of 2. That is, show that the returned vertex cover C has weight $\sum_{v \in C} w(v)$ at most twice the weight of an optimal solution.

Solution: Note that, by construction, the output C is a vertex cover and the computed dual solution is feasible. Let y be the dual solution when the primal-dual algorithm terminates. We have the following:

$$\begin{aligned}
\sum_{v \in C} w(v) &= \sum_{v \in C} \sum_{u:\{u,v\} \in E} y_{\{u,v\}} && \text{(by the definition of set } C\text{)} \\
&\leq \sum_{v \in V} \sum_{u:\{u,v\} \in E} y_{\{u,v\}} && \text{(because } y_e\text{'s are non-negative and } C \subseteq V\text{)} \\
&= 2 \cdot \sum_{e \in E} y_e && \text{(each edge is counted twice, once per each end-point)} \\
&\leq 2 \cdot LP_{OPT} && \text{(by the weak-duality theorem)} \\
&\leq 2 \cdot OPT. && \text{(because the LP is a relaxation)}
\end{aligned}$$

3 *(20 pts)* **Santa's little helper.** In Tatween, the so-called Twin World, in a galaxy far far away, Santa and the Elves are about to get busy: when we celebrate Easter, they celebrate Christmas! Here, each household has exactly one pair of twins. Since the kind Santa does not want to upset any kid, he wants to deliver **an even number of gifts per household.**

Of course there are Elves to help Santa. Each Elf knows how to make gifts for some of the households. A few days prior to Christmas, each Elf proposes to Santa the list of households for whom he is going to make gifts, along with the quantities of gifts he is making for each household. For example, an Elf might give the list *[Skywalker: 2, Vader: 1, Kenobi: 5]*, which means two gifts for the Skywalker household, one for the Vader household, and five for the Kenobi household. Santa will tell each Elf whether or not to go ahead with making the gift bundle.

Since Christmas is all about family, Santa does not want to overload any Elf family with work. Therefore **for each Elf family, he will ask at most one family member** to make his gift bundle. For example, an Elf family of three will propose three separate gift bundles, and Santa will choose at most one of these bundles to be made. The Elf families are disjoint.

Santa wants to carry some number k of gift bundles (from k different Elves). For each household, he wants to deliver the maximum possible number of gifts out of what he has. So, for example, if he caries 4 gifts for the Skywalker household, then he delivers them all, but if carries 7 gifts, then he delivers 6 of them. Remember – he does not want to upset any twin sibling by delivering an odd number of gifts to a household. (Because of this, some unfortunate households may get no gifts at all, but Santa is helpless in this case.) Notice that a single household can get gifts from many different bundles.

Despite being old, Santa is still a kid inside. He loves gifts, and after all the hard work is done, **he likes to have at least one gift remaining for himself**.

However, there is another catch. Santa's sleigh is very old and the ride is bumpy, so while flying, some of his gift bundles may fall out and get lost. In the worst case, all the bundles may fall out except for one.

Help Santa figure out (i.e., give a poly-time algorithm that finds) the maximum k such that Santa can order k bundles satisfying the following: For each Elf family, at most one member is asked to make his gift bundle, and if Santa starts his ride with those k bundles, then, no matter which bundles he loses on the way, after delivering as many gifts as possible from the remaining bundles, Santa always has at least one undelivered gift.

Example: Consider four households (Skywalker, Vader, Kenobi, Solo), six Elves of three different Elf families, and the proposed gift bundles given in the following table.

Elf family	Family member	Proposed gift bundle
Gifty-Pants	Blitz	Skywalker: 5, Kenobi: 3, Solo: 6
	Belle	Skywalker: 1, Vader: 5, Kenobi: 8
Pointy-Ears	Frost	Vader: 5, Kenobi: 7
	Faith	Skywalker: 2, Vader: 1, Kenobi: 7
Twinkle-Toes	Snow	Skywalker: 3, Vader: 4, Kenobi: 5, Solo: 2
	Sparkle	Skywalker: 7, Vader: 1

For this case, the maximum number of gift bundles that Santa can order is 2. For an example, he can ask Blitz and Sparkle to make their gift bundles, and even if one of the bags gets lost on the way, Santa will have some gifts left for himself. You can verify that it is not possible for Santa to order more than 2 bundles without violating the constraints. For example, if he orders gifts from Belle, Faith and Sparkle and loses Faith's bundle, then he will end up with 8 gifts for the Skywalkers, 6 for the Vaders and 8 for the Kenobis – all even numbers – and he will end up delivering them all.

Solution: Let n and t be the numbers of households and Elf families, respectively. We write any proposed gift bundle as a vector $v \in \{0,1\}^n$ over the finite field $GF(2)$, so that $v(i)$ represents the parity of the number of gifts proposed to household i. Thus any Elf family proposes a set of vectors. Let S_j denote the set of vectors proposed by the j-th Elf family and $E = S_1 \cup S_2 \cdots \cup S_t$ be the set of all proposed vectors.

Let F denote the set of vectors ordered by Santa. Notice that for each Elf family, Santa will ask at most one family member to make the gift bundle, hence, for any $1 \leq j \leq t$, we need $|S_j \cap F| \leq 1$. Moreover Santa likes to have at least one gift remaining for him. That is, for any non-empty subset S of F (with $F \setminus S$ being the lost bundles), the sum of vectors in S must be non-zero (the sum in $GF(2)^n$ is done coordinate-wise modulo 2). In the language of vector spaces, this means that the set F should be linearly independent. (Indeed, any nonzero combination of vectors corresponds to just taking such a subset S of F.)

Now we can solve the problem using matroid intersection. Let M_A be the partition matroid with ground set E and the partition $E = S_1 \cup S_2 \cup \cdots \cup S_t$ whose family of independent sets is given as

$$I_A = \{F \subseteq E : |F \cap S_j| \leq 1 \text{ for all } 1 \leq j \leq t\}.$$

Also let M_B be the linear matroid over $GF(2)^n$ with ground set E. Its family of independent sets M_B is

$$I_B = \{F \subseteq E : F \text{ is linearly independent}\}.$$

Since sets $F \in I_A \cap I_B$ are exactly those sets that satisfy the conditions of the problem, we need to find the size of the largest common independent set of I_A and I_B. Recall from the lecture that there is an efficient algorithm for finding a maximum-cardinality independent set in the intersection of two matroids. Therefore we can find the maximum k in polynomial time.

4 *(30 pts)* **The ad allocation problem.** We are going to investigate the linear programming relaxation of a problem that is very important for several famous tech companies. In the ad allocation problem, we are given a set A of n advertisers and a set S of m slots. Each advertiser $a \in A$ has a budget $B_a \geq 0$ and is interested in a subset $\Gamma(a) \subseteq S$ of the slots. In addition, each slot $s \in S$ has a price $p_s \geq 0$. The goal is to assign slots to advertisers so as to maximize the total profit while respecting the following constraints: (i) a slot can be assigned to at most one advertiser and (ii) the slots assigned to an advertiser a have a total price of at most B_a.

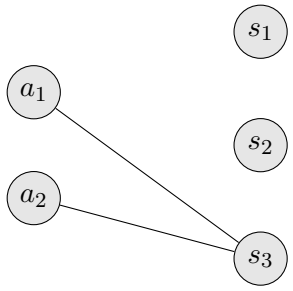
If we, for $a \in A$ and $s \in \Gamma(a)$, let x_{as} denote the indicator variable indicating that s was assigned to a, then we can formulate the ad allocation problem as the following integer linear program:

$$\begin{aligned}
\text{Maximize} \quad & \sum_{a \in A, s \in \Gamma(a)} x_{as} p_s & & \\
& \sum_{s \in \Gamma(a)} x_{as} p_s \leq B_a & \text{for all } a \in A & \quad \textit{(the budget of an advertiser should not be exceeded)} \\
& \sum_{a \in A : s \in \Gamma(a)} x_{as} \leq 1 & \text{for all } s \in S & \quad \textit{(a slot is assigned to at most one advertiser)} \\
& x_{as} \in \{0,1\} & \text{for all } a \in A,\ s \in \Gamma(a) &
\end{aligned}$$

The above integer linear program is NP-hard to solve, but we can obtain a linear programming relaxation by relaxing the constraints $x_{as} \in \{0,1\}$ to $x_{as} \in [0,1]$. The obtained linear

program can be solved in polynomial time using e.g. the ellipsoid method.

Example. An example is as follows. We have two advertisers $A = \{a_1, a_2\}$ and three slots $S = \{s_1, s_2, s_3\}$. Slot s_1 has price 1/2 and only interests a_1; slot s_2 has price 1/2 and only interests a_2; and slot s_3 has price 1 and interests both a_1 and a_2. Finally, both advertisers have unit budgets $B_{a_1} = B_{a_2} = 1$. An extreme-point solution to the linear programming relaxation is $x^*_{11} = 1$, $x^*_{22} = 1$, $x^*_{13} = 1/2$ and $x^*_{23} = 1/2$. The associated graph H (defined in subproblem **a**) can be illustrated as follows:



4a *(15 pts)* Let x^* be an extreme-point solution to the linear program and consider the (undirected) bipartite graph H associated to x^* defined as follows. Its left-hand side has a vertex u_a for each advertiser $a \in A$ and its right-hand side has a vertex v_s for each slot $s \in S$. Finally, H has an edge $\{u_a, v_s\}$ iff $0 < x^*_{as} < 1$.

Prove that H is acyclic (using that x^* is an extreme point).

Solution:

Suppose that H contains a (simple) cycle. As usual, we will show that x^* can then be written as a convex combination of two different feasible points, contradicting that x^* is an extreme point. Namely, we will define a nonzero vector y such that $x - \epsilon y$ and $x + \epsilon y$ are both feasible (for a small enough $\epsilon > 0$). Of course, $x = \frac{(x-\epsilon y)+(x+\epsilon y)}{2}$.

As in the proof for the bipartite matching polytope, label edges in the cycle alternately as odd and even. In that proof, we defined y to be $+1$ on even edges and -1 on odd edges, but this doesn't quite work here. This is because of the constraints $\sum_{s\in\Gamma(a)} x_{as}p_s \leq B_a$: the left-hand side would change when adding y because the p_s's are different.[2] To deal with this, we instead set $y_{as} = \pm 1/p_s$ for edges (a, s) in the cycle. Then:

- for each $s \in S$ we have $\sum_{a\in\Gamma(s)} y_{as} = 0$ because this sum has two nonzero terms, equal to $1/p_s$ and $-1/p_s$ (or no nonzero terms if s does not appear in the cycle),
- for each $a \in A$ we have $\sum_{s\in S:a\in\Gamma(s)} y_{as}p_s = 0$ because this sum has two nonzero terms, equal to $1/p_s \cdot p_s$ and $-1/p_{s'} \cdot p_{s'}$ where s and s' are the neighbors of a on the cycle (or no nonzero terms if a does not appear in the cycle),
- since $y_{as} \neq 0$ only if $0 < x^*_{as} < 1$, we can set ϵ so that $0 \leq x^*_{as} \pm \epsilon y_{as} \leq 1$ for all a and s.

This shows that the points $x \pm \epsilon y$ are both feasible.

[2] It is not enough to set ϵ small enough. Consider an instance with two advertisers and two slots. The slots have prices 1 and 2, and both advertisers have budget 1.5. Let x assign 1/2 of each slot to each advertiser. Then just adding/subtracting $\pm\epsilon$ on the cycle-edges will violate one of the advertiser's budgets, no matter what ϵ is.

4b *(15 pts)* **You may assume the structural property of a) even if you did not solve that subproblem.** Use the structural result proved in the first subproblem to devise an efficient rounding algorithm that, given an instance and a feasible extreme point x^* in the linear programming relaxation corresponding to the instance, returns an ad allocation of total profit at least

$$(1-\gamma) \sum_{a \in A, s \in \Gamma(a)} x^*_{as} p_s \qquad \text{where } \gamma := \max_{a \in A, s \in \Gamma(a)} \frac{p_s}{B_a}\,.$$

Notice that your algorithm gives a very good solution assuming that the budget of each advertiser is much larger than the price of any slot that he is interested in. This is a reasonable assumption (online advertisers buy thousands/millions of slots) and is used in the online primal-dual algorithms that form a basis of what runs in practice.

Solution: Let us first do the "obvious" thing and assign a slot s to an advertiser a whenever we have $x^*_{as} = 1$. Surprisingly, it turns out that just doing that already gives the above profit guarantee! The trick is to realize/prove this...

Intuition. What to do with the fractional edges? We build the graph H out of them. Since H is acyclic, it is a forest; say it is just a single tree. A tree has leaves. Imagine we have an advertiser a that is a leaf. What can we do with it?

Since a is a leaf, it has exactly one slot s that is fractionally assigned to it. Can we integrally assign s to a without violating the budget of a? If yes, then it looks like a good idea to do so. (Actually, for our profit bound we do not need to, but since it increases the profit, then why not?) Then both a and s disappear from H (i.e., they become isolated vertices) and we are still making (at least) as much profit as we were previously, since s is used.

If we cannot assign s to a without violating the budget of a, then we give up on s. (Again, to get our profit bound, we can always give up on s.) We make the same progress as before in terms of H, since a and s become isolated. However, now we lose $p_s \cdot \sum_{a \in A} x^*_{as} \leq p_s$ in profit. We "blame" a for this loss.

Then we continue the same procedure with another leaf-advertiser.

To analyze the profit, notice that each advertiser a is blamed only for at most one slot $s \in \Gamma(a)$. Thus a is blamed for the loss of at most a $\frac{p_s}{B_a} \leq \gamma$ fraction of its budget. This is good enough – at least if a was a *tight* vertex (i.e., its corresponding constraint in the LP is tight). In that case, it is blamed for the loss of at most a γ fraction of its *LP value*. Clearly, at most a γ fraction of the total LP value can be lost in this way.

To conclude, we need to show that we can choose a tight leaf-advertiser at each step (until H becomes empty).

Full solution. Assume there are no slots s with $\sum_a x^*_{as} = 0$ (we can disregard them). For any slot s that the LP assigns to a single advertiser a, i.e., $x^*_{as} = 1$, it makes sense to listen to the LP and assign s to a. Now we argue that doing this is enough to get the wanted total profit. For this, we construct an injection from fractional slots (those that appear in H as non-isolated vertices) to tight advertisers. To do this, we use the graph H, which has the following properties:

1. H is acyclic, i.e., a forest.

2. In each connected component of H there is at most one non-tight vertex. (A vertex, either an advertiser or a slot, is tight if its corresponding constraint is tight.)

3. Slot-leaves (i.e., leaves of H that are slots) are non-tight.

Let us argue for each of them:

1. Proved in subproblem a).
2. Assume towards a contradiction that H contains a path connecting two non-tight vertices. Then we can repeat the proof from subproblem a), moving the edges of this path by $\pm\epsilon/p_s$. All vertices except for the endpoints of this path are fine, since the sums in the corresponding constraints do not change (just like for the cycle). Regarding the endpoints, by choosing ϵ small enough we can ensure that their corresponding constraints are not violated (since they were not tight in x^*).
3. The corresponding constraint is $\sum_{a\in A} x^*_{as} \leq 1$. It is in fact a strict inequality, because the left-hand side is made up by a single edge, which is fractional.

Now, while H contains an edge, we do the following:

- Choose a tight advertiser-leaf a. Let its (only) neighbor be s.

  Such an advertiser exists because H is nonempty, so it has a connected component that is a nonempty tree. This tree has at least two leaves. Since at most one of them is non-tight (property 2), there is a tight leaf. By property 3, it must be an advertiser.

- We can now either assign s to a if that is possible within the budget B_a, or not. Say we do not. This causes a loss (in that the profit of our solution is smaller than the LP value) of at most p_s. "Blame" a for the loss of s.

- We remove a and s from H. Note that this preserves all of the above properties of H (in particular, no slot ever becomes a leaf in H).

Obviously, this procedure will terminate. It remains to argue that the total profit is high. For each $a \in A$, let $LP_a = \sum_{s\in\Gamma(a)} x^*_{as} p_s$ be the profit that the LP gets from a. The LP objective value is $\sum_a LP_a$. If a is tight, then $LP_a = B_a$.

Let S_B be the set of "bad" slots – those that were not assigned to any advertiser. For $s \in S_B$ define $adv(s) \in A$ to be the advertiser a that is blamed for the loss of s. The function $adv : S_B \to A$ is one-to-one. Moreover, by the definition of γ and tightness of $adv(s)$ we have

$$p_s \leq \gamma \cdot B_{adv(s)} = \gamma \cdot LP_{adv(s)} \,. \tag{1}$$

Our total profit is $\sum_{s\in S} p_s - \sum_{s\in S_B} p_s$. Clearly, the first sum is at least the LP objective value. By (1) the second sum is at most $\sum_{s\in S_B} \gamma \cdot LP_{adv(s)} \leq \gamma \cdot \sum_{a\in A} LP_a$, i.e., γ times the LP objective value. This yields that the total profit is at least $(1-\gamma)$ times the objective value.

5 *(20 pts)* **Implementation.** The objective of this problem is to successfully solve the problem *Hiking Trails* on our online judge. You will find detailed instructions on how to do this on Moodle.

5a *(17 pts)* Get an Accepted verdict on the online judge.

Solution: Note that this is the weighted vertex cover problem (huts are vertices, trails are edges). One algorithm that can be used is given in Problem 2 (the budgets correspond to the dual values).

Listing 1: Implementation of the primal-dual algorithm for weighted vertex cover in Python 3

```
def get_ints(): # read a line of integers as an array
    return [int(x) for x in input().split(' ')]

# read input
n, m = get_ints()
c = get_ints()
trails = [get_ints() for i in range(m)]
budgets = [0] * m
# c[i] will serve as slack of the dual constraint corresponding to i-th hut/
    vertex (at first it is the cost, since budgets are zero)

# primal-dual algorithm
for i, (u, v) in enumerate(trails): # for every edge
    if c[u - 1] > 0 and c[v - 1] > 0: # that is not covered
        budgets[i] = min(c[u - 1], c[v - 1]) # increase budget as much as
            possible
        c[v - 1] -= budgets[i] # update slack of constraint
        c[u - 1] -= budgets[i] # update slack of constraint

# print output; we take those huts whose slacks have been brought down to 0
    (tight vertices)
print(c.count(0))
for i in range(n):
    if c[i] == 0:
        print(i+1, end=' ')
print()
print(*budgets)
```

5b *(3 pts)* On Moodle we have provided you with some of the testcases for the problem. For each of them, say `04`, we also give a corresponding file `04.opt`, which contains two numbers: the optimum value of the natural linear programming relaxation for that instance, and the (integral) optimum value for that instance. In your solution, produce a table with one row per testcase, containing:

- the testcase name (e.g. `04`),
- the approximation ratio that your algorithm can guarantee for itself (i.e. the ratio of the cost of its solution to the sum of the budgets from the problem statement),
- the approximation ratio that your algorithm gets when compared to the LP value (i.e. the ratio of the cost of its solution to the LP value given in `04.opt`),
- the approximation ratio that your algorithm gets when compared to the true optimum value (i.e. the ratio of the cost of its solution to the integral optimum value given in `04.opt`).

Are these results roughly in line with what you had expected? (You do not need to answer this question in your solution but it is good to think about.)

Solution: See the ratios in Table 1. As we (should have) expected, we have the following inequalities:

$$2 \geq \text{apx wrt found dual} \geq \text{apx wrt } \mathrm{OPT}_{\mathrm{LP}} \geq \text{apx wrt OPT} \geq 1\,.$$

The more interesting question is how tight these inequalities are. We can see that some instances are solved to optimality and the solution can certify this using the found dual values (budgets)

testcase	apx wrt found dual	apx wrt $\mathrm{OPT}_{\mathrm{LP}}$	apx wrt OPT
01	1.5	1.5	1.5
02	1.6	1.44	1.0
03	1.0	1.0	1.0
04	1.0	1.0	1.0
05	1.68	1.57	1.11
06	1.9	1.84	1.21
07	1.87	1.79	1.19
08	1.87	1.82	1.12
09	1.93	1.84	1.17
10	1.87	1.79	1.1
11	1.73	1.71	1.71
12	1.88	1.87	1.87
13	1.87	1.87	1.87
14	1.95	1.9	1.52
15	1.89	1.82	1.27
16	1.74	1.5	1.14
17	1.88	1.73	1.14
18	1.77	1.5	1.15
19	1.28	1.07	1.07
20	1.7	1.55	1.15

Table 1: The approximation ratios obtained by a primal-dual algorithm. (The theoretical guarantee is 2 – see Problem 2.)

(see e.g. testcase 03); others are solved to optimality (or close), but the solution finds it hard to certify a low ratio even if were to solve the LP "on the side" (e.g. 02); and some testcases are rather far off (e.g. 12). We remark that our testcases were generated more or less randomly; "real-world" instances might exhibit additional structure that would make them easier to solve using approximation algorithms.