Problem 1: Min-Cut (20 points)

(this problem is due to Emo Welzl)

Let G denote a weighted graph with n vertices and positive weights on edges. Let μ be the weight of a minimum cut in G (i.e. the sum of the weights of all the edges in the cut).

(a) Show that the total weight of G (i.e. the sum of weights of all the edges in G) is at least $\mu n/2$.

Let $G = (V, E)$, and let $w : E \to \mathbb{R}_+$ be the weight function. For every vertex $u \in V$ define weight $w(u)$ as $\sum_{(u,v)\in E} w((u, v))$. Clearly, $w(u) \geq \mu$, $\forall\, u \in V$ (otherwise there would exist a cut with weight smaller than μ). Hence we get, using the fact that every edge is adjacent to two vertices,

$$\begin{aligned} \sum_{e\in E} w(e) &= \frac{1}{2}\sum_{u\in V}\sum_{(u,v)\in E} w((u,v)) \\ &= \frac{1}{2}\sum_{u\in V} w(v) \\ &\geq \frac{1}{2}\sum_{u\in V} \mu = \mu n/2\ . \end{aligned}$$

□

(b) Is the lower bound from part (a) the best possible? In other words, is it true that for every $n > 1$ and $\mu > 0$ there exists a weighted graph G with positive weights, such that a minimum cut of G has value μ and the total weight of G is $\mu n/2$?

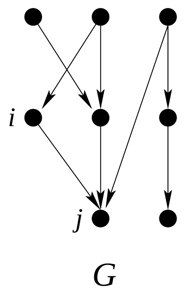
Yes, the lower bound from part (a) is the best possible. For a given $n > 1$ and $\mu > 0$ consider $K_n = (V, E)$, a complete graph with n vertices, in which every edge has weight $\mu/(n-1)$. Any cut partitioning V into two sets V_1 and V_2, with $|V_1| = k$ and $|V_2| = n-k$, cuts $k(n-k)$ edges. Hence the minimum is reached by the cuts separating just one vertex ($k = 1$ or $k = n-1$), i.e. the minimum cut has cost μ. Moreover, graph K_n has $n(n-1)/2$ edges, so its total weight is $\mu n/2$. □

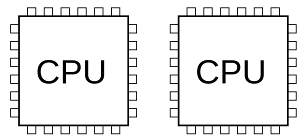
(c) Is the assumption, that the weights are positive, essential for the assertion of part (a)?

No, the assumption is not essential — in the proof of part (a) we make no use of this assumption.

Problem 2: 2-Processor Scheduling (30 points)

Assume you are given a collection of n jobs, each requiring a unit processing time, and two identical processors. The jobs are arranged as vertices of a directed acyclic (precedence) graph G: an edge (i, j) in G means that job i must be finished before j is started (see an example below)





Give an efficient algorithm, which finds a schedule for the two processors, which completes all the jobs in the shortest possible time.

Prove the correctness of your algorithm and analyze its time complexity.

Hints:

1. Think maximum matching.
2. Don't be intimidated, if the matching you've found doesn't directly yield the desired schedule — maybe it is still the case that this matching can help you find the schedule.

ALGORITHM We call two jobs i and j *compatible*, if in G there is no directed path from i to j or from j to i (i.e. i and j can be executed in parallel). Consider the following algorithm:

1. From G construct an undirected *compatibility* graph H, which has the same set of vertices as G, and there is an (undirected) edge (i, j) iff jobs i and j are compatible.
2. Find a maximum matching M in graph H.
3. Create a schedule based on M — every job matched by M runs in parallel with some job from M, and every non-matched job runs separately, with the other processor idle:

   Let S be the subset of vertices with of G with in-degree equal 0. Repeat the following steps until all the jobs are scheduled

   (a) if there is an unmatched vertex in S, schedule it and remove it from G

   (b) if there exists a pair of matched jobs and both jobs are in S, schedule this pair and remove both jobs from G

   (c) if neither (a) nor (b) is applicable, find two pairs of matched jobs (i, j) and (i', j') such that $i \in S$, $i' \in S$ and j and j' are compatible (this is possible, see the proof of correctness below); replace the two matched pairs by (i, i') and (j, j'), and continue (now there exists a matched pair with both jobs in S).

CORRECTNESS First we show that the algorithm finds a correct schedule. Since the algorithm schedules only compatible jobs, it is sufficient to show that it won't "get stuck". If neither (a) nor (b) is applicable, then all jobs in S are matched, and each job in S is matched with a job not in S. Moreover, since G is acyclic, there must exist at least two matched pairs, say (i, j) and (i', j'), such that one job of each pair is in S and the other is not in S. The two jobs not in S, j and j', can be compatible or not. If they are, we are done, otherwise, if wlog. j' precedes j in G, there must exist another pair (i'', j''), with $i'' \in S$ and $j'' \notin S$, such that j and j'' are compatible — otherwise we could construct a cycle or an infinite path of jobs preceding j, which is impossible, since G is acyclic and finite.

Now we show that the algorithm finds the shortest possible schedule. Let m be the number of edges in a maximum matching of H. All the matched jobs are run in pairs (possibly different from the pairs of the original matching), and the $(n - 2m)$ non-matched jobs are executed on a single processor, with the other processor idle. Hence the total length of the schedule is $m + n - 2m = n - m$. To see that this is the shortest possible schedule, observe that any schedule S yields naturally a matching in H — just match the jobs which run in parallel in S. A schedule shorter than the one found by the above algorithm would imply the existence of a matching larger than the maximum matching, which is impossible.

COMPLEXITY The construction of H in step 1 can be implemented by taking the complement of the transitive closure of G, which takes in total $\mathcal{O}(n^3)$. Step 2 takes $\mathcal{O}(m\sqrt{n})$ time (using the Micali-Vazirani algorithm), where m is the number of edges in H. Finally, each iteration of Step 3 can be implemented to take $\mathcal{O}(n^2)$ time, and there are $\mathcal{O}(n)$ iterations. Hence the total running time of the algorithm is bounded by $\mathcal{O}(n^3)$.