CSE 202 Homework #4

1. **Problem 1: Hamiltonian path**

   High-level Description

   Assume that $n \geq 2$, otherwise there is always a Hamiltonian path and the problem is trivial.

   We present a quite complicated algorithm to check the existence of Hamiltonian path from v_1 to v_n in G in $O(2^n n^3 \log n)$ time and $O(n^2)$ space.

   We first design a helper function PathCount(V', E'), which counts the number of paths from v_1 to v_n of length n. Here the "path" means a sequence of vertices in which each two adjacent vertices has a directed edge from the former to the latter, and "length" means the number of vertices in the sequence.

   The algorithmic content in the helper function is as follows:

   - We construct an adjacency matrix $A[1..n][1..n]$, where $A[i][j] = 1$ if and only if $v_i, v_j \in V'$ and there is an directed edge from v_i to v_j in E' (otherwise $A[i][j] = 0$).
   - We calculate $B = A^{n-1}$ by matrix multiplication.
   - We return $B[1][n]$ as the answer.

   Then we can use this helper function to find the number of Hamiltonian paths from v_1 to v_n. Let $V_{cand} = \{v_2, \cdots v_{n-1}\}$, as V_{cand} has 2^{n-2} subsets, we call PathCount 2^{n-2} times. Denote $E(V')$ as the subset of E in which all the edges have both the incoming vertex and outgoing vertex in V'. When we are enumerating the subset of V_{cand} as V'_{cand}, we call PathCount$(V'_{cand} + \{v_1, v_n\}, E(V'_{cand} + \{v_1, v_n\}))$ and denote the return value as $pc(V'_{cand})$. After we have all the 2^{n-2} $pc(V'_{cand})$ entries, we can calculate the number of Hamiltonian paths from v_1 to v_n by the following equation:

   $$\#\text{ of Hamiltonian paths } = \sum_{V'_{cand} \subset V'_{cand}} (-1)^{n-|V'_{cand}|} \cdot pc(V'_{cand}) \tag{1}$$

   If the number of Hamiltonian paths is larger than 0, we can claim that there is a path in G that goes from v_1 to v_n passing through every other vertex exactly once, otherwise there is no such path.

   Proof of Correctness

Claim 1. *PathCount*(V', E') *correctly returns the number of paths from* v_1 *to* v_n *of length* n.

Proof. We can prove a stronger statement (i.e., $B[v_i][v_j]$ is the number of paths from v_i to v_j) by induction on the length of the path p.

Base Case The length of the path is at least 2 since it must including the starting and ending vertex. When $p = 2$, we have $B = A^1 = A$ the same as the original adjacency matrix. Since there is a path of length 2 from v_i to v_j if and only if there is a direct edge from v_i to v_j, the matrix B contains the correct information.

Induction Step For $p > 2$, if $B' = A^{(p-1)-1}$ contains the correct information about the paths of length $p-1$, for $B = A^{p-1} = B'A$ and every to vertices v_i and v_j, the number of paths from v_i to v_j of length p can be calculated as follows: we enumerate the second-last vertex in the path (denoted as v_k), if there is a direct edge from v_k to v_j, we can add the number of paths from v_i to v_k of length $p-1$ to the answer. Based on the definition of matrix multiplication:

$$B[i][j] = \sum_{k=1}^{n} B'[i][k] \times A[k][j]$$

since $B'[i][k]$ is actually number of paths from v_i to v_k of length $p-1$ and $A[k][j]$ indicates whether there is a direct edge from v_k to v_j, we prove the correctness of B for p. □

Claim 2. *Equation 1 correctly counts the number of Hamiltonian paths from* v_1 *to* v_n.

Proof. For every Hamiltonian path from v_1 to v_n, it must contain every vertex $v_1, \cdots, v_n$ by definition, thus it is counted only when we are calling PathCount by $V'_{cand} = \{v_2, \cdots, v_{n-1}\}$. The coefficient for this V'_{cand} in Equation 1 is $(-1)^{n-(n-2)} = 1$, thus every Hamiltonian path correctly contributes 1 to the answer.

For every non-Hamiltonian path from v_1 to v_n, denote the set of vertices in this path as V_p. It is counted for every V'_{cand} that satisfies $V_p \subset V'_{cand}$, thus it contributes:

$$\sum_{V_p \subset V'_{cand}} (-1)^{n-|V'_{cand}|} \tag{2}$$

to the answer.

The V'_{cand} entries has the same contribution if they have the same cardinality. There are $\binom{n-2-|V_p|}{k-|V_p|}$ entries that has cardinality k and contribute $(-1)^{n-k}$ to Equation 2. Thus Equation 2 can be reformulated as:

$$\begin{aligned}\sum_{k=|V_p|}^{n-2}\binom{n-2-|V_p|}{k-|V_p|}\times(-1)^{n-k} &= \sum_{k'=0}^{n-2-|V_p|}\binom{n-2-|V_p|}{k'}\times(-1)^{n-k'-|V_p|}\\ &= \sum_{k'=0}^{n-2-|V_p|}\binom{n-2-|V_p|}{k'}\times(-1)^{(n-2-|V_p|)-k'}\\ &= (1-1)^{n-2-|V_p|}=0\end{aligned}$$

which means every non-Hamiltonian path contributes 0 to the answer.

Thus the equation 1 can be used to calculate the number of Hamiltonian paths. □

Time Complexity

We need to call the helper function $2^{n-2} = O(2^n)$ times, during which the construct of A takes $O(n^2)$ time and the calculation of $B = A^{n-1}$ takes $O(n^3 \log n)$ time (if we apply binary exponentiation). Thus the total time complexity is $O(2^n \cdot n^3 \log n)$, which is $\in O(2^n \cdot \text{poly}(n))$.

Space Complexity

We need $O(n^2)$ space to store A, B and some intermediate representations when calculating B. The return value (i.e. $pc(V'_{cand})$) is no need to store explicitly because it is only used to calculate one entry in the summation of Equation 1. Thus the total space complexity is $O(n^2)$, which is $\in O(\text{poly}(n))$

Reference

Karp, R.M., 1982. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2), pp.49-51.

2. **Problem 2: Heaviest first**

(1) For each node $v \in T$, if $v \notin S$, it is deleted by another node v' and $v' \in S$, which indicates that (v, v') in an edge of G. We claim that $w(v) \leq w(v')$ and it can be proved by contradiction.

If $w(v) > w(v')$, as we are doing a "heaviest-first" greedy algorithm, at the time we are picking v', v' is the node that has the largest weight, thus v cannot be remained in G, which contradicts the assumption that v is deleted by v'.

(2) Let S be the independent set returned by the "heaviest-first" greedy algorithm, and T be any other independent set in G. We say that every $v \in T$ is "connected" to a node v' in S based on (1):

- Either $v \in S$, we have $v' = v$. $v \in T$ is connected to itself.
- Or $v \notin S$, we have $v' \in S$ which satisfies $w(v) \leq w(v')$ and v, v' are neighbors in G. $v \in T$ is connected to $v' \in S$.

On the other head, every $v' \in S$ is connected to at most 4 nodes in T. This is because v' can only connect to itself and its (at most) 4 neighbor nodes, but as the 5 nodes cannot appear in T according to the independent set restriction, v' will have at most 4 connected nodes in T.

Denote $c(v)$ $(v \in T)$ as the node in S which is connected to T. As each $v' \in S$ appears at most 4 times in all $c(v)$ $(v \in T)$ entries, we have:

$$\begin{aligned}\sum_{v \in T} w(v) &\leq \sum_{v \in T} w(c(v)) \\ &\leq \sum_{v' \in S} 4w(v) \\ &= 4 \sum_{v' \in S} w(v')\end{aligned}$$

which means that $\dfrac{\sum_{v' \in S} w(v')}{\sum_{v \in T} w(v)} \geq \dfrac{1}{4}$ holds. Thus it proves the problem statement.

3. **Problem 3: Scheduling**

(1) For example, there are 7 jobs and the time requirements are $[6, 5, 4, 3, 2, 2, 2]$.

The greedy heuristic would result in the scheduling $[6, 3, 2, 2]$ and $[5, 4, 2]$, where the total time is $\max(13, 11) = 13$.

But we can arrange the jobs as $[6, 2, 2, 2]$ and $[5, 4, 3]$, where the total time is 12.

Thus the greedy heuristic is not optimal.

(2) According to the problem statement, we have $t_1 \geq t_2 \geq \cdots \geq t_n$. We first prove that when n is small, the greedy heuristic is always optimal.

Claim 3. *When $n \leq 4$, the greedy heuristic is always optimal.*

Proof. We can prove this by enumerating all possible n values.

- When $i = 1$, the greedy heuristic will assign the only job to a machine, which is the only valid assignment and it is optimal.
- When $i = 2$, the greedy heuristic will assign one job to each machine, which is the only valid assignment and it is optimal.
- When $i = 3$, the greedy heuristic will assign job 1 to one machine and job $2, 3$ to the other. We can show that this is optimal, because for the other 2 valid assignments $(1, 2), (3)$ or $(1, 3), (2)$:

$$\begin{cases} \max(t_1 + t_2, t_3) = t_1 + t_2 \geq \max(t_1, t_2 + t_3) \\ \max(t_1 + t_3, t_2) = t_1 + t_3 \geq \max(t_1, t_2 + t_3) \end{cases}$$

  Here we ignore the trivial bad case where all the jobs are assigned to one machine.
- When $i = 4$, if $t_1 \geq t_2 + t_3$, the greedy heuristic will assign job 1 to one machine and job $2, 3, 4$ to the other. This is optimal because if job 1 is together with any other job (say job x), as $t_1 + t_x > t_1$ and $t_1 + t_x \geq t_2 + t_3 + t_x \geq t_2 + t_3 + t_4$ holds, the assignment cannot be better. If $t_1 < t_2 + t_3$, the greedy heuristic will assign job $1, 4$ to one machine and job $2, 3$ to the other. We can show that this is optimal, because for the other 6 valid assignments $(1), (2, 3, 4)$, $(1, 3, 4), (2)$, $(1, 2, 4), (3)$, $(1, 2, 3), (4)$, $(1, 2), (3, 4)$, $(1, 3), (2, 4)$:

$$\begin{cases} \max(t_1, t_2 + t_3 + t_4) = t_2 + t_3 + t_4 \geq \max(t_1 + t_4, t_2 + t_3) \\ \max(t_1 + t_3 + t_4, t_2) = t_1 + t_3 + t_4 \geq \max(t_1 + t_4, t_2 + t_3) \\ \max(t_1 + t_2 + t_4, t_3) = t_1 + t_2 + t_4 \geq \max(t_1 + t_4, t_2 + t_3) \\ \max(t_1 + t_2 + t_3, t_4) = t_1 + t_2 + t_3 \geq \max(t_1 + t_4, t_2 + t_3) \\ \max(t_1 + t_2, t_3 + t_4) = t_1 + t_2 \geq \max(t_1 + t_4, t_2 + t_3) \\ \max(t_1 + t_3, t_2 + t_4) = t_1 + t_3 \geq \max(t_1 + t_4, t_2 + t_3) \end{cases}$$

  Here we ignore the trivial bad case where all the jobs are assigned to one machine.

□

We can prove that when n is large, the greedy heuristic has an approximation ratio of $\frac{7}{6}$.

Claim 4. *When $n > 4$, the greedy heuristic has an approximation ratio of $\frac{7}{6}$.*

Proof. Denote the answer from greedy heuristic as T_{GRE} and the optimal answer and T_{OPT}. Without loss of generality, we assume that the completion time of the first machine is strictly larger than the second machine (if the completion time are the same, we split the jobs into two groups of the same completion time, which indicates a perfect optimal assignment). Denote the last job assigned to the first machine as r and it needs t_r time. If $r < n$, we can ignore all the jobs $r+1, r+2, \cdots, n$, because they are assigned to the second machine and will not affect the completion time (because the first machine has larger completion time). By ignoring these jobs, T_{GRE} does not change while T_{OPT} would decrease, indicating that $\frac{T_{GRE}}{T_{OPT}}$ increases and we are trying to prove a stronger case.
Thus we can safely assume that $r = n$. Denote the completion time of the two machines as T_1 ($T_1 = T_{GRE}$) and T_2, we have $T_1 > T_2$ and $T_1 - t_n \leq T_2$, which means:

$$T_1 - t_n \leq \frac{1}{2}((T_1 - t_n) + T_2) = \frac{1}{2}\sum_{i=1}^{n-1} t_i$$

$$\Rightarrow \quad T_1 \leq \frac{1}{2}\sum_{i=1}^{n} t_i + \frac{1}{2}t_n$$

For T_{OPT} we have:

$$T_{OPT} \geq \frac{1}{2}\sum_{i=1}^{n} t_i$$

Then we can prove by contradiction. If $\frac{T_{GRE}}{T_{OPT}} > \frac{7}{6}$, we will have:

$$\frac{7}{6} < \frac{T_{GRE}}{T_{OPT}} \leq \frac{\frac{1}{2}\sum_{i=1}^{n} t_i + \frac{1}{2}t_n}{T_{OPT}} \leq \frac{T_{OPT} + \frac{1}{2}t_n}{T_{OPT}} = 1 + \frac{t_n}{2T_{OPT}}$$

$$\Rightarrow \quad t_n > (\frac{7}{6} - 1) \cdot 2T_{OPT} = \frac{1}{3}T_{OPT}$$

But $t_n > \frac{1}{3}T_{OPT}$ infers that $t_1, t_2, \cdots, t_{n-1} > \frac{1}{3}T_{OPT}$, thus each machine could be assigned with **at most** 2 jobs in the optimal solution, indicating that $n \leq 4$, which contradicts our assumption that $n > 4$.

□

Reference

Graham, R.L., 1969. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2), pp.416-429.

(3) **Problem 4: Maximum coverage**

Denote $w_1, \cdots, w_n$ as the weight of the n elements, $U_{GRE,i}$ as the set of elements of our greedy approach after picking i subsets, U_{OPT} as the set of elements of the optimal solution. Denote $U_{TAR,i} = U_{OPT} \backslash U_{GRE,i}$ as the set of elements that are picked by the optimal solution but not our greedy approach after picking i subsets.

At the time when we have $U_{GRE,i-1}$ and $U_{TAR,i-1}$ and are picking an i-th subset:

- If $U_{TAR,i-1} = \emptyset$, we have covered all the elements in the optimal solution, indicating that we achieve a perfect solution by greedy approach.
- If $U_{TAR,i-1} \neq \emptyset$, we claim that we will pick a subset in which the total weights of elements besides $U_{GRE,i-1}$ is at least $\frac{1}{k}$ of the total weights of the elements in $U_{TAR,i-1}$ (i.e. $\sum_{j \in U_{TAR,i-1}} w_j$).

  Proof. The optimal solution covers all the elements in $U_{TAR,i-1}$ and it has k subsets, at least one of the subset must satisfy the restriction that the total weights of elements it covers in $U_{TAR,i-1}$ is at least $\frac{1}{k} \sum_{j \in U_{TAR,i-1}} w_j$. Denote this subset as S_{o_i}. S_{o_i} is of course not selected in our greedy approach, because it covers the elements in $U_{TAR,i-1}$ and by definition $U_{TAR,i-1}$ is the set of elements that our greedy approach hasn't covered.

  As $U_{TAR,i-1} = U_{OPT} \backslash U_{GRE,i-1} \in U \backslash U_{GRE,i-1}$, when we are picking an i-th subset during our greedy approach, we will pick a subset no worse than S_{o_i}, in which the total weights of elements besides $U_{GRE,i-1}$ is at least $\frac{1}{k}$ of the total weights of the elements in $U_{TAR,i-1}$. □

Let $U_{GRE,0} = \emptyset$, if our greedy approach does not meet a perfect solution on the halfway, we will have:

$$\begin{aligned}
\frac{\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,i}} w_j}{\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,i-1}} w_j} &= 1 - \frac{\sum_{j \in U_{GRE,i}} w_j - \sum_{j \in U_{GRE,i-1}} w_j}{\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,i-1}} w_j} \\
&\leq 1 - \frac{\frac{1}{k} \sum_{j \in U_{TAR,i-1}} w_j}{\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,i-1}} w_j} \\
&= 1 - \frac{\frac{1}{k} \sum_{j \in U_{OPT} \backslash U_{GRE,i-1}} w_j}{\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,i-1}} w_j} \leq 1 - \frac{1}{k}
\end{aligned}$$

thus:

$$\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,k}} w_j \leq (1 - \frac{1}{k})^k \left(\sum_{j \in U_{OPT}} w_j - \sum_{j \in U_{GRE,0}} w_j \right) = (1 - \frac{1}{k})^k \sum_{j \in U_{OPT}} w_j$$

which indicates the approximate factor $\frac{\sum_{j \in U_{GRE,k}} w_j}{\sum_{j \in U_{OPT}} w_j} \geq 1 - (1 - \frac{1}{k})^k > (1 - \frac{1}{e})$.