Design and Analysis of Algorithms (Autumn 2023), 5 cr

Course exam 25.10.2012 – Model solutions

Examiner: Veli Mäkinen

Remember to write your name on each answer sheet. You can answer also in Finnish or in Swedish.

Course grading: 15+15+15+15=60. The assignment yielding least points will be replaced by exercise points if that gives a higher score.

1. **Recurrences**

   Consider recurrence $T(n) = 2T(n/2) + n \log_2 n$. The following *flawed* substitution proof aims to prove $T(n) = O(n \log^2 n)$. Spot the errors in the proof and then fix them to give a correct analysis. Before fixing the induction proof, check first using the recursion tree method if $O(n \log^2 n)$ is the correct bound.

   Flawed analysis.

   Assume $T(m) \leq cm \log_2^2 m$ for $m < n$.

   Then by induction

   $$\begin{aligned} T(n) &\leq 2cn/2 \log_2^2 \tfrac{n}{2} + n \log_2 n \\ &\leq cn \log_2^2 n + n \log_2 n \\ &= O(n \log^2 n). \end{aligned}$$

   Solution.

   First, the recursion tree yields $T(n) = n \sum_{i=0}^{\log_2 n} \log \frac{n}{2^i} < n \sum_{i=0}^{\log_2 n} \log n = O(n \log^2 n)$, so the bound is indeed correct, just the analysis is flawed.

   The analysis estimates the equation too much upwards failing to show that $T(n) \leq cn \log_2^2 n$ as it should. Also base case is missing. More careful analysis is sufficient to fix the flaws.

   Assume again $T(m) \leq cm \log_2^2 m$ for $m < n$.

   Then by induction

   $$\begin{aligned} T(n) &\leq 2cn/2 \log_2^2 \tfrac{n}{2} + n \log_2 n \\ &= cn(\log_2^2 n - 2 \log_2 n + 1) + n \log_2 n \\ &\leq cn \log_2^2 n, \end{aligned}$$

   when $-2cn \log_2 n + cn + n \log_2 n \leq 0$. This holds for $c \geq 1$ and $n \geq 2$, as then $cn \leq cn \log_2 n$.

   Assume the natural base case $T(1) = 0$. Induction assumes $T(1) \leq c1 \log_2^2 1 = 0$, which holds in the base case.

   Grading.

   - 5 points for checking that the bound is correct using the recursion tree method.
   - 7 points for fixing the induction step correctly.
   - 3 points for fixing the constants by checking the base case.

   Common mistake.

   A common mistake was to subtract a lower order term, e.g., $T(m) \leq cm \log_2^2 m - bm \log_2 m$ for $m < n$, but showing only $T(n) \leq cn \log^2 n$, while one should show $T(n) \leq cn \log^2 n - bn \log_2 n$ in this case.

2. **Flows**

A *cycle cover* of a directed graph $G = (V, E)$ is a set of vertex-disjoint simple cycles that cover all the vertices, i.e., every vertex appears exactly once in the cycle cover. In the Cycle Cover problem we are asked whether G has a cycle cover or not.

Show that the Cycle Cover problem can be solved in polynomial time by reducing it to the maximum-flow problem. Remember to provide a complete proof. *Hint.* Start with a bipartite graph with one copy of nodes on the left and another copy on the right.

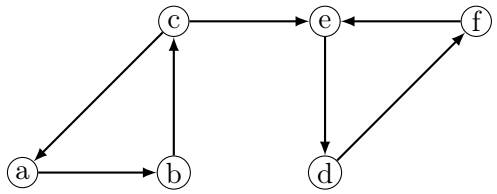
Solution.

(a) We want to reduce the Cycle Cover problem to Maximum-flow problem. Thus we take an input of the Cycle Cover problem, i.e a graph $G = (V, E)$, and we create an input of Maximum-flow problem. We create two different copies of V, V_{out} and V_{in}, we take $G' = (V', E')$ the directed graph with

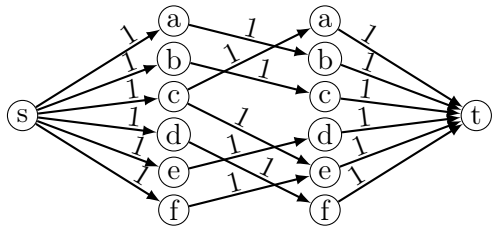
$$\begin{aligned} V' &= \{s\} \cup V_{out} \cup V_{in} \cup \{t\} \\ E' &= (\{s\} \times V_{out}) \bigcup (V_{out} \times V_{in} \cap E) \bigcup (V_{in} \times \{t\}) \end{aligned}$$

where s and t are new nodes corresponding to the *source* and the *target* and the capacity c with for all $e' \in E'$, $c(e') = 1$.

G



G'



As $|V'| = 2|V| + 2$ and $|E'| = 2|V| + |E|$, the size of G' is in $O(|V| + |E|)$.

Now we want to prove that G has a cycle cover **if and only if** G' has a feasible flow of size $|V|$ with only integral values.

(b) ($\Rightarrow$) Assume that G has a cycle cover C represented as a set of arcs defining the cycles. As a cycle cover is formed of vertex-disjoint simples cycles, we can decompose C in a set of vertex-disjoint simples cycles $\{c_1, \ldots, c_m\}$. As $\{c_1, \ldots, c_m\}$ is vertex-disjoint, we can partition V in $V_1, \ldots, V_m$ using the set of vertex cover by each cycle. As for each V_i the number of vertices in V_i corresponds to the number of arcs in c_i, the total number of arcs is equal to the number of vertices in V, i.e. $|C| = |V|$.

We take the application f from E' to $\mathbb{N}$ which is initialized to 0 and for each $(u, v) \in C$, we add 1 to $f((s, u))$, $f((u, v))$ and $f((v, t))$. As for each $(u, v) \in C$ we create a path from s to t, f is a flow (respects the flow convervation property).

As C is formed of vertex-disjoint simple cycles, for all (u, v) and $(u', v') \in C$, we have $u \neq u'$ and $v \neq v'$. Hence, we can prove by contradiction that for all $e' \in E'$, $f(e') \in \{0, 1\}$. Indeed, we assume that there exists (u, v) such that $f((u, v)) \geq 2$. As E is not a multiset (G has no multiple edges between the same pair of vertices), we know that if $(u, v) \in E$, $f((u, v)) \in \{0, 1\}$, thus it is impossible. If $u = s$, there exists

w and w' such that (v, w) and $(v, w') \in C$ which is impossible. If $v = t$, there exists w and w' such that (w, u) and $(w', u) \in C$ which is also impossible. Hence, the flow f is feasible.

As for all $e' \in E'$, $f(e') \in \{0, 1\}$, the size of f corresponds to the number of $u \in V_{out}$ such that $f((s, u)) = 1$, which corresponds to the number of arcs of C. As $|C| = |V|$, the size of f is $|V|$. Hence there exists a feasible flow of size $|V|$ with only integral values.

(c) ($\Leftarrow$) Assume that G' has a feasible flow f of size $|V|$ with only integral values. As f is a feasible flow with only integral values, we have that for all $e' \in E'$, $f(e') \in \{0, 1\}$. As the size of f is $|V|$, $|\{(s, u) \in E'\}| = |V_{out}|$ and $|V_{out}| = |V|$, for each vertex u of V_{out}, we have $f((s, u)) = 1$. As f is a flow and satisfies the flow conservation property, for each $u \in V_{out}$, there exists a unique $(u, v) \in E$ such that $f(u, v) = 1$. We denote by C the following set of $|V|$ elements $\{(u, v) : u \in V_{out}, (u, v) \in E \text{ and } f(u, v) = 1\}$. We are going to prove that C is a cyclic cover of G. As for all $u \in V_{out}$ there exists $(u, v) \in C$ and V_{out} is a copy of V, C covers all the vertices of V. By the definition of C, for each $u \in V$, there exists a unique v such that $(u, v) \in C$ and as $|V| = |C|$, C is a set of cycles and thus a cycle cover of G.

Grading. 5 points for each item. In detail

(a) **1 point** for taking an input of Cycle Cover, **1 point** for creating a flow network, **1 point** for giving a good network and **2 points** to give the good equivalence between the problems.

(b) **1 point** for taking a cyclic cover, **1 point** to create a flow with this input, **1 point** to prove that is a flow and **2 points** to prove that this flow is feasible.

(c) **1 point** for taking a flow with the good properties, **1 point** to create a cyclic cover with the flow, **1 point** to prove that cover G and **2 points** to prove that is a set of cycles.

3. **Dynamic programming**

Recall the algorithm to compute the length of the longest increasing subsequence in quadratic time. Modify the algorithm so that it computes the length of the longest increasing subsequence that can be partitioned to k pieces that can be found in that order as subarrays of the input. For example, with input $A = 1, 3, 5, 2, 6, 7, 2, 4, 9, 12, 15$, and $k = 3$, the longest increasing subsequence $= 1, 3, 5, 6, 7, 9, 12, 15$ is also the solution to this restricted case as it can be partitioned into $[1, 3, 5], [6, 7], [9, 12, 15]$, where the pieces are found in that order in A. With $k = 2$ the optimal solution is $[1, 3, 5], [9, 12, 15]$, and thus differs from the unrestricted optimum. *Hint.* Consider computing a two-dimensional table $T[0 \ldots n][0 \ldots k]$ with value $T[i][j]$ defined as the length of the longest increasing subsequence in the subarray $A[1..i]$ consisting of j contiguous pieces such that the last piece ends in the ith element, or $-\infty$ if no such subsequence exists. Find a recurrence to compute the values in a suitable evaluation order. Consider also initialization and finalization.

Solution.

We solve the problem using dynamic programming, computing for all $0 \leq i \leq n$ and $0 \leq j \leq k$ the value $T[i][j]$ defined as the length of the longest increasing subsequence in the subarray $A[1..i]$ consisting of j contiguous pieces such that the last piece ends in the ith element, or $-\infty$ if no such subsequence exists. For convenience, we define that if there are no pieces, the last piece ends in $A[0] = -\infty$. By this definition, the final result is given by $\max_{0 \leq i \leq n} T[i][k]$.

From the definition we get that $T[0][0] = 0$ and if one of the equalities $i = 0$ or $j = 0$ holds, then $T[i][j] = -\infty$. Let us now compute $T[i][j]$ for $1 \leq i \leq n$ and $1 \leq j \leq k$ recursively.

Constructing an increasing subsequence consisting of j contiguous pieces ending in $A[i]$ can be done in two ways: extending the last piece of a subsequence with j contiguous pieces ending in $A[i-1]$ provided that $A[i-1] < A[i]$ or by creating a new piece into an increasing subsequence constisting of $j-1$ contiguous pieces ending in $A[a]$ where $0 \leq a < i$ and $A[a] < A[i]$. If $A[i-1] \geq A[i]$, we get the recurrence

$$T[i][j] = \max_{\substack{0 \leq a < i \\ A[a] < A[i]}} T[a][j-1] + 1$$

and if $A[i-1] < A[i]$, we get the recurrence

$$T[i][j] = \max \left\{ T[i-1][j] + 1, \max_{\substack{0 \leq a < i \\ A[a] < A[i]}} T[a][j-1] + 1 \right\}.$$

Here we use the convention that a maximum over an empty set is $-\infty$. By definition, the value $T[i][0]$ is 0 if $i = 0$ and $-\infty$ otherwise. The straightforward implementation of this recurrence runs in $O(kn^2)$.

Grading.

- 8 points from the correct recurrence.
- 5 points from the correct initialization.
- 2 points from the correct finalization.

4. **NP-completeness of vertex cover revisited**

In this assignment you should complete an NP-completeness proof of vertex cover using the following sketch of a reduction from 3-CNF-SAT. In *vertex cover* problem, one asks if there is a subset of nodes of size k in an undirected graph that cover all edges. That is, each edge should have at least one endpoint inside the subset forming a cover. Recall that we already proved vertex cover NP-complete, but for this assignment, assume you do not know it yet.

Recall also that the input of *3-CNF-SAT* problem is a boolean formula of the following form:

$$\phi = (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge (\ell_{2,1} \vee \ell_{2,2} \vee \ell_{2,3}) \wedge \cdots \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m,3}),$$

where each literal $\ell_{i,j}$ is either x_t or $\neg x_t$, for some $t > 0$, and $\{x_1, x_2, \ldots\}$ is the set of variables that may occur in ϕ. Recall that we proved that deciding if 3-CNF-SAT formula ϕ has a truth assignment is an NP-complete problem, and for this assignment, you can assume it known.

Consider a reduction that creates a graph with the $3m$ literals as nodes. Add edges to connect all literals inside the same clause to each other. Then for each literal add also an edge to its negation in all other clauses. (Note that this is the complement graph of the one we considered at the lecture for another problem.) That is, for

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

we would create a graph with six nodes $v_{1,1}, v_{1,2}, v_{1,3}, v_{2,1}, v_{2,2}, v_{2,3}$ corresponding to the literals in the two clauses in the same order. Edges would be $(v_{1,1}, v_{1,2}), (v_{1,1}, v_{1,3}), (v_{1,2}, v_{1,3}), (v_{2,1}, v_{2,2}), (v_{2,1}, v_{2,3}), (v_{2,2}, v_{2,3}), (v_{1,1}, v_{2,1}), (v_{1,3}, v_{2,3})$.

Complete the details of this reduction to prove vertex cover NP-complete. *Hint.* Think of vertex cover of size $k = 2m$.

Solution.

Vertex cover belongs to NP because it admits a certificate of the following form: the set S of vertices in the cover. To check such a certificate in polynomial time, we inspect each edge of G and check whether one of its endpoints belongs to S.

To prove that vertex cover is NP-hard, observe that the reduction proposed can be done in polynomial time. We still need to prove that ϕ is satisfiable $\Leftrightarrow$ G (i.e., the graph proposed in the reduction) has a vertex cover of size $2m$.

"$\Rightarrow$": Let $f(\cdot)$ be an assignment of truth values to the variables of ϕ (i.e., $f(x_t)$ is either 0 or 1) rendering ϕ true. For each clause c_i of ϕ, let $\ell_{i,j}$ be one literal in c_i set to true by f. We select in the cover S the two nodes of c_i *different* from $\ell_{i,j}$. Observe that the size of S is $2m$. The set S is also a vertex cover because:

- all edges inside the same clause are covered by S, as we select two nodes from each clause;
- any possible edge between clauses c_i and c_j is of the form $(x_t, \neg x_t)$. Thus, at least one of its endpoints is selected in the cover for c_i or for c_j (i.e., not both of x_t and $\neg x_t$ were set to true by f).

"$\Leftarrow$": Let S be a vertex cover of G of size $2m$. Observe that any vertex cover of G must select at least 2 nodes from each clause. Since $|S| = 2m$, then S selects exactly 2 nodes from each clause. We construct a truth assignment f by setting to true that literal of c_i not in S, for each c_i. If consistent (i.e., we do not assign true to both literals x_t and $\neg x_t$), then f renders true each clause, and thus renders ϕ satisfiable. To see that it is consistent, assume for a contradiction that both x_t and $\neg x_t$ were assigned to true in distinct clauses c_i and c_j, respectively. Thus, neither literal x_t or $\neg x_t$ is in S. Thus the edge $(x_t, \neg x_t)$ is not covered by S, a contradiction.

Grading.

- 1 point: arguing that vertex cover belongs to NP and that the reduction given in the assignment works in polynomial time.
- 2 points: mentioning the equivalence ϕ is satisfiable $\Leftrightarrow$ G has a vertex cover of size $2m$ (either explicitly or implicitly)
- 6 points: proving the "$\Rightarrow$" implication
- 6 points: proving the "$\Leftarrow$" implication