AuW-u02-bf

ETH Zürich
Institute of Theoretical Computer Science
Prof. Rasmus Kyng
Prof. Angelika Steger

FS 2024

Algorithmen und Wahrscheinlichkeit
Theorie-Aufgaben 2

Abgabe in Moodle () bis zum 21.03.2024 um 10:00 Uhr.

Aufgabe 1 – *Jass-Karten*

Ein Jass-Kartenspiel besteht aus 36 Karten mit vier Farben (Rosen, Schellen, Eichel, Schilten) und neun Werten (6, 7, 8, 9, 10, Under, Ober, König, Ass). Die Karten werden gemischt und zufällig auf 9 Stapel $S_1, \ldots, S_9$ mit je vier Karten aufgeteilt. Sie dürfen sich nun aus jedem Stapel eine Karte aussuchen. Können Sie die Wahl so treffen, dass Sie am Ende eine vollständige Strasse ausgewählt haben (also jeden der 9 Kartenwerte genau einmal)?

(a) Finden Sie einen Algorithmus, der als Input $S_1, \ldots, S_9$ (wie oben beschrieben) nimmt, und als Output angibt, ob eine solche Wahl möglich ist.

(b) Geben Sie auch einen (effizienten) Algorithmus an, der eine Lösung berechnet. (Der also ausgibt, welche Karte Sie von welchem Stapel auswählen sollen, um eine vollständige Strasse zu erhalten.)

Hinweis: Obwohl die Fragestellungen scheinbar nichts mit Graphen zu tun haben, lässt sich das Problem als graphentheoretisches Problem formulieren. Geben Sie eine solche Formulierung an. Dazu müssen Sie insbesondere genau beschreiben, welchen Graphen $G = (V, E)$ Sie betrachten. (Was ist V? Was ist E?) Anschliessend können Sie beschreiben, was Sie über G wissen, und was Ihnen dieses Wissen über die ursprüngliche Fragestellung verrät. Dafür dürfen Sie in der Vorlesung beschriebene Algorithmen zitieren, selbst wenn dort die Implementierung nicht besprochen wurde. Achten Sie jedoch darauf, dass die Algorithmen in der Vorlesung nur für einfache Graphen beschrieben wurden.

a)

For the purpose of simplicity, let (1) be the mapping of numerical values corresponding to the card values. Note, that this mapping is arbitrarily chosen and does not influence the outcome. Let (2) be a function returning every cards numerical value. Let (3) be some auxiliary definitions and (4) the definition of the bipartite Graph G, representing the stacks of playing cards and the values they contain.

$$
\begin{aligned}
&(1)\\
&1 \simeq \text{”6”}\\
&2 \simeq \text{”7”}\\
&3 \simeq \text{”8”}\\
&4 \simeq \text{”9”}\\
&5 \simeq \text{”10”}\\
&6 \simeq \text{”Under”}\\
&7 \simeq \text{”Ober”}\\
&8 \simeq \text{”König”}\\
&9 \simeq \text{”Ass”}
\end{aligned}
$$

$$
\begin{aligned}
&(2)\\
&\nu(k) := w, \quad k = (w, h) \in \mathcal{K}
\end{aligned}
$$

$$
\begin{aligned}
&(3)\\
&\mathcal{I} = \{1, 2, .., 4\}\\
&\mathcal{J} = \{1, 2, .., 9\}\\
&\mathcal{W} = \mathcal{J}\\
&\mathcal{H} = \{\diamondsuit, \clubsuit, \heartsuit, \spadesuit\}\\
&\mathcal{K} = \mathcal{W} \times \mathcal{H}\\
&\mathcal{S}_{j \in \mathcal{J}} = \{k_{i \in \mathcal{I}} \in \mathcal{K} \mid k_i \notin \mathcal{S}_{j' \in \mathcal{J} \setminus \{j\}}\}
\end{aligned}
$$

$$
\begin{aligned}
&(4)\\
&G = (V, E)\\
&A = \{\mathcal{S}_j \mid j \in \mathcal{J}\}\\
&B = \mathcal{J}\\
&V = A \uplus B\\
&E = \{(\mathcal{S}_{j \in \mathcal{J}}, w) \in A \times B \mid \exists k \in \mathcal{S}_j : \ \nu(k) = w\}
\end{aligned}
$$

Let G be the bipartite graph representing the exclusive union of the partition of stacks of cards and let an edge between a stack of cards and a value denote, that there exists (at least) a card of said value in the aforementioned stack.

Construction of the graph takes $\mathcal{O}(36) = \mathcal{O}(1)$ time. The algorithm is

```
return true
```

Which is again, $\mathcal{O}(1)$.
□

---

b)

A straightforward approach is to iterate over all possible combinations, which, given the constant number of cards, takes $\mathcal{O}(1)$:

```
// pick-cards(stack S1, S2, ... S9)

picked = {}
for stack Si do
        card ci_fmax = {}
        for card cj in stack Si
                if f(cj) > f(ci_fmax) do
                        if !picked.contains(val(cj)) do
                                picked.add(cj)
                        elseif !picked.contains(val(ci_fmax)) do
                                picked.add(ci_fmax)
                        else backtrack() end
                end
        end
end

function backtrack()
        picked.remove(#picked)    // remove card last added
        go back to last decision
        if !pick-different-possible do backtrack()
        else pick-different end
end

function pick different()
        pick a different card and continue
end

print(picked)
```

A more efficient approach is to use the fact, that G is bipartite (let the two partitions be the stacks and card values). We can use the Hopcroft-Karp algorithm to find a perfect matching (which exists, since G is bipartite). We pick all cards in the perfect matching's vertex set.

Runtime analysis

Graph construction: $\mathcal{O}(36) = \mathcal{O}(1)$

Hopcroft-Karp algorithm: $\mathcal{O}(\sqrt{|V|} \cdot (|V| + |E|)) \leq \mathcal{O}(\sqrt{36} \cdot (36 + 18)) = \mathcal{O}(1)$

Thus the algorithm has constant runtime.

□