AuW-u02-bf

ETH Zürich FS 2024
Institute of Theoretical Computer Science
Prof. Rasmus Kyng
Prof. Angelika Steger

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)

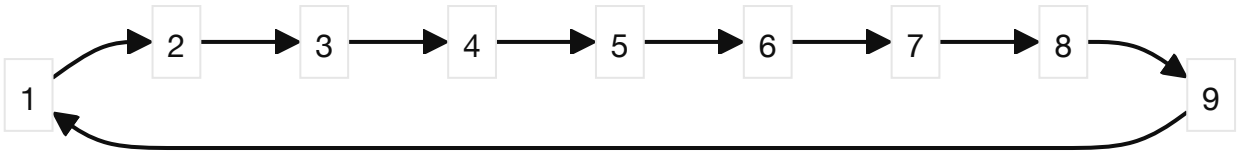
Let $G = (V, E)$ be the connected 4-regular Graph representing the playing cards.
For the purpose of simplicity, let

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

be the mapping of numerical values to card values. Note, that this mapping is arbitrarily chosen and does not influence the outcome. Thus,

$$
\begin{aligned}
V &= \{\mathbb{Z}_9^* \times \{\diamondsuit, \clubsuit, \heartsuit, \spadesuit\}\} \\
E &= \{(v_1, v_2) \in V \times V \mid v_1 \prec v_2\}
\end{aligned}
$$

where the function $\nu : V \to \mathbb{Z}_9^*$, maps every card to its numerical value and the relation $\prec \overset{def}{=} \nu(v_1) + 1 \equiv_9 \nu(v_2)$ on V defines, if a given card comes one before another. Note, that we assume the card values to loop back on themselves.



The algorithm is thus:

```
// straight-possible(stack S1, S2, ... S9)

map cards = {}
for stack Si do
        for card cj in stack Si do
                cards.put(cj.suit, cj.value)
        end
end

if cards.keys().size() != 4 print("No") end    // not four suits
for suit in cards.keys() do
        if suit.size() != 9 print("No") end   // not 9 cards per suit
end

print("Yes")    // deck complete => straight possible
```

$

---

b)

Let $G' = (V', E')$ and let $S_1, S_2, \dots S_9$ as described. Furthermore, let $f : V \to \mathbb{Z}_4^*$ denote, how often we have seen a given card's numerical value.
We construct the graph as follows:

$$V' = \bigcup_{i \in \mathbb{Z}_9^*} v \in S_i : f_{\max}(v), \forall v' \in V' \, \nu(v) \neq \nu(v')$$
$$V' \implies E'$$

By construction, the graph contains one card from every stack, that together form a straight (Strasse).

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

print(picked)
```

A more efficient approach is to use the fact, that G is 4-regular and bipartite (let the two partitions be the cards with even numerical value and the cards with odd numerical value). We can use the Hopcroft-Karp algorithm to find a perfect matching (which exists, since G is regular and connected). We pick all cards in the perfect matchings vertex set.

An even more efficient approach is to use the fact, that G is 4-regular and bipartite (let the two partitions be the cards with even numerical value and the cards with odd numerical value). We can use the following algorithm:

```
// linear-pick-cards(G)

E = euler-tour(G)     // euler-tour of G
2F = every-other(E)   // 2-factor of G
M1 = every-other(2F)   // perfect matching of G, colors = {a, b}
M2 = every-other(2F)   // perfect matching of G, colors = {c, d}, M1 ≠ M2

P = 2-color(M1) xor 2-color(M2)   // 4-colored set of edges
picked = 1-color set in P
print(picked)

function euler-tour(G)
        return euler tour of G
end

function every-other(G)
        E' = remove every other edge from E
        return G = (V, E')
end

function 2-color(G)
        for edge in E alternate color
end
```