Karlsruher Institut für Technologie

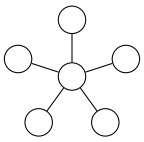
Lösung zum Übungsblatt 5

Abgabe bis 15. Januar 2025

Aufgabe 1: Sterne

8 Punkte

Ein *5-Stern* ist folgender Graph:



Gegeben einen ungerichteten Graphen $G = (V,E)$ und Parameter k soll entschieden werden, ob G mindestens k knotendisjunkte induzierte 5-Sterne enthält. Verwende *color coding* um zu zeigen, dass dieses Problem in FPT liegt.

Lösung 1:

Gegeben einen ungerichteten Graphen $G = (V, E)$ und Parameter k ist die Frage, ob G (mindestens) k knotendisjunkte induzierte 5-Sterne enthält.

Hierfür betrachten wir zunächst das Colorful Stars Problem. Hier liegt für einen gegebenen Graphen zusätzlich noch eine Färbung vor und es muss für die Sterne zusätzlich gelten, dass alle $6k$ Sternknoten paarweise verschiedene Farben haben. Wenn wir für dieses Problem einen FPT Algorithmus finden, so erhalten wir mithilfe einer $(n, 6k)$-perfekten Familie von Hashfunktionen auch einen FPT Algorithmus für das ursprüngliche Problem.

Da beim Colorful Stars Problem alle Sternknoten eine andere Farbe haben müssen, hat auch jeder Stern 6 verschiedene Farben und die Farben von zwei unterschiedlichen Sternen sind allesamt unterschiedlich. Wir probieren somit alle potentiellen Zuordnungen von Farben zu Sternen durch. Da es nur $6k$ Farben und k Sterne gibt, hängt die Anzahl Kombinationen hierfür nur von k ab. Eine grobe Abschätzung liefert $O((6k)!)$. Für jede Zuordnung schauen wir uns dann für jeden der k Sterne mit zugehöriger Farbmenge alle 6-Tupel von Knoten an ($O(n^6)$ viele) und prüfen ob die 6 Knoten richtig gefärbt sind und als Teilgraph isomorph zu einem 5-Stern sind. Falls wir für eine Zuordnung für alle k Sterne eine entsprechende Knotenmenge finden, können wir „Ja“ ausgeben, sonst „Nein“. Die Disjunktheit der Sterne wird durch die Färbung garantiert.

Laufzeit: Eine $(n, 6k)$ perfekte Familie von Hashfunktion finden wir in $O(n6.46k)$. Insgesamt ergibt sich für den FPT-Algorithmus also eine Laufzeit von $O(6.46k^2 \cdot (6k)! \cdot n^7)$.

Aufgabe 2: Longest Cycle **12 Punkte**

Gegeben sei ein Graph $G = (V, E)$ und ein Parameter k. Bei Longest Cycle geht es darum, zu entscheiden ob es in G einen Kreis der Länge *mindestens* k in G gibt. Gib einen FPT-Algorithmus für dieses Problem an.

Hinweis: Beachte, dass es Graphen gibt, die keinen Kreis der Länge genau k enthalten, aber dennoch einen Kreis der Länge mindestens k.

Lösung 2:

Im Folgenden möchten wir einen FPT-Algorithmus angeben, der für einen Graph $G = (V, E)$ und Parameter k entscheidet, ob G einen induzierten Kreis der Länge mindestens k enthält.

Um uns das Problem zu erleichtern, möchten wir zunächst einen Algorithmus entwickeln, der entscheidet ob es einen induzierten Kreis der Länge genau k gibt. Hierfür überprüfen wir für jede Kante $\{u, v\} \in E$, ob es zwischen u und v einen induzierten Pfad der Länge genau k gibt. Dazu können wir den Algorithmus zum suchen bunter Pfade aus der Vorlesung verwenden, indem wir u und v jeweils mit Blättern u' und v' verbinden, welchen wir einzigartige Farben geben, sodass jeder bunte Pfad der Länge $k + 2$ definitiv u' und v' verwenden muss.

Wir gehen nun wie folgt vor: für jedes i zwischen k und $2k$ überprüfen wir mit dem obigen Algorithmus, ob G einen Kreis der Länge i enthält. Ist dies der Fall, können wir Ja ausgeben, andernfalls kontrahieren wir eine beliebige Kante und setzen unsere Suche im resultierenden Graphen so lange fort, bis dieser nur noch konstant viele Kanten enthält. Dies ist korrekt, da sich die Länge des längsten Kreises durch die Kontraktion einer Kante höchstens halbieren kann. Falls G also einen Kreis der Länge $\ell > 2k$ enthält, so kontrahiert unser Algorithmus so lange Kanten, bis die Länge des Kreises zwischen k und $2k$ liegt (oder ein anderer ausreichend langer Kreis gefunden wird).

Aufgabe 3: Dominating Set auf speziellen Graphen **10 Punkte**

Betrachte folgende parametrisierte Variante von Dominating Set. Der Graph G ist gegeben zusammen mit einer Knotenordnung $V = \{v_1, \dots, v_n\}$. Die *Länge* einer Kante $\{v_i, v_j\}$ ist $|i - j|$. Der Parameter k ist die maximale Kantenlänge in G. Gib einen FPT-Algorithmus an, der diese Parametrisierung von Dominating Set löst.

Lösung 3:

Gegeben ist ein Graph $G = (V,E)$ mit einer Knotenordnung $V = \{v_1, \dots, v_n\}$, wobei für alle Kanten $\{v_i, v_j\} \in E$ gilt, dass $|i - j| \leq k$. Wir stellen fest, dass wir eine Pfadzerlegung der Weite $2k$ erhalten, indem wir jeweils $2k$ aufeinanderfolgenden Knoten in einen gemeinsamen Bag stecken.

Mithilfe der Pfadzerlegung können wir Dominating Set (DS) mit einem dynamischen Programm lösen. Für jeden Bag X speichern wir uns Teillösungen die für jeden Knoten $x \in X$ angeben, ob

dieser im DS enthalten ist, von einem anderen Knoten abgedeckt wird oder nicht im DS enthalten und nicht abgedeckt ist. Außerdem speichern wir für jede Teillösung die bisherige Größe des konstruierten DS. Für den ersten Bag der Pfadzerlegung können wir alle 2^{2k} Möglichkeiten Knoten ins DS hinzuzufügen durchprobieren.

Wir müssen uns nun überlegen, wie wir Knoten zu Teillösungen hinzufügen und von Teillösungen löschen. Wenn ein Knoten hinzugefügt wird, kann dieser zum DS hinzugefügt werden oder nicht, wodurch aus jeder alten Teillösung zwei neue entstehen, für die wir gegebenenfalls die Größe und den Abgedecktheits-Status der enthaltenen Knoten aktualisieren müssen. Beim Entfernen eines Knotens v, müssen wir Teillösungen als ungültig markieren, falls v nicht im DS ist und auch nicht als abgedeckt markiert ist.

Aufgrund der Konstruktion der Pfadzerlegung können wir immer abwechselnd Knoten hinzufügen und entfernen, ohne jemals Bags mit mehr als $2k$ Knoten betrachten zu müssen. Da jeder Knoten nur auf drei verschiedene Arten markiert sein kann, ergeben sich pro Bag höchstens 3^{2k} viele Teillösungen. Insgesamt ergibt sich also eine Laufzeit in $6^k \cdot n^{O(1)}$.

Aufgabe 4: Closest String

5 Bonus-Punkte

In dieser Aufgabe sollst du erneut ein Programm implementieren, das das Problem Closest String (in einer leichten Variante) löst. Gegeben sind k Strings der Länge n. Gesucht ist ein minimales D, sodass ein String s^* existiert, der zu jedem anderen String Hamming-Distanz höchstens D hat.

Löse das Problem mittels eines ILP wie in der 3. Übung. Du darfst dir gerne weitere Optimierungen überlegen. Beschreibe in der PDF-Abgabe, welche Änderungen du an der ILP Formulierung aus der Vorlesunge vorgenommen hast und ob es sonst noch Einsichten gab, die dir beim Lösen von dem Problem geholfen haben. Gib außerdem in der PDF-Abgabe für jede Instanz das kleinste D an, für das du eine Lösung gefunden hast.

Gib zusätzlich den Quellcode sowie deine gefundenen Lösungen (im unten beschriebenen Format) als eine ZIP-Datei ab. Für jede gelöste Instanz kannst du einen Punkt bekommen.

Dateiformat Eingabe: In der ersten Zeile steht k, die Anzahl der Strings. In den nächsten k Zeilen ist jeweils ein String gegeben. Alle Strings haben die selbe Länge und verwenden nur die kleinen Buchstaben `a-z`.

Dateiformat Ausgabe: Eine Zeile mit einem String s^*, der minimale Hamming-Distanz zu den gegebenen Strings hat.

Hinweis: Mithilfe der Datei `validator.py` kannst du die Hamming-Distanz deiner Lösung zu den gegebenen Strings bestimmen. Dabei wird das Maximum über alle Hamming-Distanzen ausgegeben.

Hinweis: Für diese Aufgabe benötigst du einen ILP-Solver. Die Anzahl der Programmiersprachen und Solver ist groß, aber du solltest in der Lage sein, die Aufgabe mit den meisten Kombinationen

zu lösen. Einer der mit Abstand besten Solver ist `Gurobi`. Um ihn benutzen zu können, brauchst du allerdings eine Lizenz, die du kostenlos mit deiner Studenten-E-Mail erhältst.

Lösung 4:

String Instanz	**Minimales** D
Instanz 0	72
Instanz 1	731
Instanz 2	871
Instanz 3	8380
Instanz 4	86835

Wir wünschen euch schöne Weihnachtsferien!