Prof. Dr. Heinz Schmitz
Fachbereich Informatik
Hochschule Trier

Algorithmen-Design
Sommersemester 2019
20. März 2019

Inhalt

2.4. Repräsentation im Rechner
Adjazenzmatrix, Adjazenzmengen, Speicherplatz, Vorteile Adjazenzmengen, Standardrepräsentation, Graphen in PYTHON.

3. Traversierung von Graphen

Entscheidungsproblem GRAPH REACHABILITY (GR).

3.1. Breitensuche
Idee, induktive Definition der Schichten L_i, `bfs(G,s)`, Eigenschaften des Algorithmus, BFS-Baum, Implementierung, Laufzeit $O(m+k)$, Varianten.

3.2. Tiefensuche
Idee, rekursiver Algorithmus `dfs_rec(G,u)`, DFS-Baum, iterative Implementierung `dfs(G,u)` mit Laufzeit $O(m+k)$, Vergleich BFS / DFS.

3.3. Zusammenhangskomponenten
Berechnung aller Zusammenhangskomponenten für ungerichtete Graphen in $O(m+k)$, Umkehrgraph G^{rev}, Berechnung der starken Zusammenhangskomponenten in Digraphen.

4. Lösungsräume und Lösungsbäume

4.1. Modellierung mit Mengen und Prädikaten
Damenproblem, Modellierung, Definition $sol(m)$, Entscheidungsproblem QUEEN; Rucksackproblem, Modellierung, Optimierungsproblem MAXIMUM KNAPSACK .

4.2. Durchmustern von Lösungsräumen
Systematisches Erzeugen aller Lösungskandidaten, Voraussetzungen, Entwurfsmuster *Exhaustive Search*, Laufzeit, Anwendung `queen_exhaustive`.

4.3. Lösungsräume mit Baumstruktur
Idee, Vergleich Knotenanzahl mit *Exhaustive Search*; Idee Backtracking-Kriterium K, Entwurfsmuster *Backtracking*, Varianten, Anwendung `queen_backtracking`; Idee oberer Schranken, Definition, Entwurfsmuster *Branch and Bound*, obere Schranke für MAXIMUM KNAPSACK.

5. Greedy-Algorithmen 1: Scheduling

Entwurfsmuster *Greedy*, *Greedy*–Auswahl, Konsequenzen.

5.1. Die beste Belegung für einen Raum
Problembeschreibung Raumbelegung, kompatible Räume, Modellierung als MAX INTERVAL SCHEDULING, Alternativen für die Greedy-Auswahl,

Satz: Algorithmus `max_intsched_greedy` liefert optimale Lösung in $O(m \log m)$.

5.2. Wer zu spät kommt ...
Problembeschreibung, Deadlines d_i und Lateness l_i, Modellierung als MINIMUM LATENESS SCHEDULING, Greedy-Auswahl *Earliest Deadline First*, Satz: Es gibt einen optimalen Schedule ohne Inversionen und ohne Leerlaufzeiten.

6. Greedy-Algorithmen 2: Kürzeste Wege und minimale Spannbäume

6.1. Der Algorithmus von Dijkstra
Kürzeste Wege in Digraphen, SHORTEST PATH; *Greedy*–Entwurf nach Dijkstra, Aktualisierung der Schätzwerte für $d(u)$, `dijkstra`, kürzeste Pfade P_u, Korrektheitssatz, Laufzeitverbesserung mit Min Priority Queue, Satz: `dijkstra` kann in $O(k \log m)$ implementiert werden.

6.2. Der Algorithmus von Prim
Minimale Spannbäume, *Greedy*–Entwurf nach Prim, Aktualisierung der Schätzwerte für $d(u)$, `prim`, Beweis der Schnitteigenschaft, Korrektheitssatz, Laufzeitverbesserung mit Min Priority Queue, Satz: `prim` kann in $O(k \log m)$ implementiert werden.

7. Greedy-Algorithmen 3: Nochmal Spannbäume

7.1. Der Algorithmus von Kruskal
Greedy–Entwurf nach Kruskal, `kruskal`, Korrektheitssatz.

7.2. Die *UnionFind*–Datenstruktur
Verwaltung inkrementeller Zusammenhangskomponenten, Spezifikation von $makeUnionFind(m)$, $find(i)$ und $union(A, B)$; Implementierung mit Mengen `sets[j]` und Liste `set_containing[i]`, Klasse `UnionFind`, Laufzeit; Verwendung in `kruskal`, Satz: `kruskal` kann mit der $UnionFind$-Datenstruktur in $O(k \log m)$ implementiert werden.

7.3. Effiziente Clusterung
Problembeschreibung, Distanzfunktion $d(p_i, p_j)$, l-Clusterung, Abstand, Modellierung von MAXIMUM l-CLUSTERING, Algorithmenentwurf mit `kruskal`-Variante in $O(m^2 \log m)$, Korrektheitssatz.

8. Divide and Conquer

   Rekursive Lösungsbeschreibung, optimale Teilstrukturen, Bellmansches Optimalitätsprinzip.

   8.1. Entwurfsmuster und Beispiele
   Beispiel `merge_sort`, Entwurfsmuster *Divide and Conquer*, Laufzeitanalyse `merge_sort`, geschlossene Form durch Ausrollen der Rekursion, Funktion f mit $f(m) \leq 2f(m/2) + cm$ ist in $O(m \log m)$, Beispiel fib-Funktion, überlappende Teilaufgaben, Entwurfsmuster *Memoization*.

   8.2. Das Punktepaar mit kleinstem Abstand
   Problembeschreibung, euklidische Distanz $d(x, y)$, Algorithmus `cp_all_pairs` in $O(m^2)$, Modellierung CLOSEST PAIR, zusätzliche Annahmen; Algorithmenentwurf nach *Divide and Conquer*, 1-dim. Variante, Auswahl der Datenstruktur, Divide-Schritt, Conquer-Schritt in $O(m)$ durch Absuchen des δ-Korridors, Algorithmus `cp_dac` (Gerüst), Satz: `cp_dac` löst CP optimal in $O(m \log m)$.

9. Rekursionsgleichungen

   Allgemeine Rekursionsgleichungen.

   9.1. Lösungsverfahren
   Ausrollen, zulässige Vernachlässigungen, Methode des induktiven Einsetzens (3 Schritte), konstruktive Induktion, das Master-Theorem und seine Anwendung.

   9.2. Schnelle Multiplikation nach Karatsuba
   Schulmethode in $O(m^2)$, Teilungsidee und Analyse, Verbesserung auf drei Teilprobleme, Satz: `karatsuba` multipliziert zwei m-Bit Zahlen in $O(m^{1,59})$.

10. Dynamische Programmierung 1: Scheduling

    10.1. Intervall-Scheduling mit Gewichten
    MAX WEIGHTED INTERVAL SCHEDULING, maximal zulässiges Vorgängerintervall $p(j)$, Analyse einer optimalen Lösung mit $m - 1 \in A^{opt}$ und $m - 1 \notin A^{opt}$, *cut & paste*–Argumente, Teilprobleme x_j, Rekusionsgleichung für optimale Werte $opt(j)$, nicht-rekursive Berechnung, Satz: `max_weighted_intsched_dp` löst MAX WEIGHTED INTERVAL SCHEDULING in $O(m \log m)$; Konstruktion optimaler Lösungen, Mehrdeutigkeiten.

10.2. Ein elementares Packproblem
MAX SUM OF SUBSETS, Analyse von I^{opt}, erweiterte Teilprobleme (x_j, s) mit zusätzlicher Variable, optimale Werte $opt(j, s)$.

11. Dynamische Programmierung 2: Kürzeste Wege

11.1. Prinzipien der Dynamischen Programmierung
Vorgehensweise zur Bestimmung einer rekursiven Lösungsbeschreibung, Widerspruchsbeweis mit *cut& paste*–Argumenten; algorithmische Umsetzung mit Muster *Dynamic Programming*, d.h. tabellarische *bottom-up*-Berechnung aller Teillösungen, Vergleich mit *Memoization*.

11.2. Der Algorithmus von Bellman und Ford
Negative Kantenkosten und negative Kreise, `dijkstra`–Gegenbeispiel, INTEGER SHORTEST PATH, Herleitung einer rekusiven Lösungsbeschreibung, Implementierungsaspekte von `bellman_ford`, Satz: `bellman_ford` löst INT SHORTEST PATH in Zeit $O(mk)$ und Speicherplatz $O(m)$; Konstruktion kürzester Pfade aus dem *Bellman-Ford*–Graph G_{BF}; Überblick über verwandte Algorithmen.

12. Flussnetzwerke 1: Grundlagen

12.1. Modellierung mit Flussnetzwerken
Flussnetzwerk G mit Quelle s und Senke t, s-t-Fluss f, Kapazitätsbedingung, Flusserhalt, Flusswert $v(f)$, zusätzliche Annahmen, MAX FLOW, *Greedy*-Gegenbeispiel.

12.2. Der Algorithmus von Ford und Fulkerson
Idee Rückwärtskanten, Residualgraph G_f mit Vorwärts- und Rückwärtskanten sowie Restkapazitäten r_e, Konstruktion mit `residual_graph` in $O(m + k)$, *Bottleneck*-Wert, vergrößernde Pfade in G_f mit Tiefensuche, `augment` liefert Fluss in G, Algorithmus `ford_fulkerson`; Terminierungsargumente, Satz: `ford_fulkerson` hat Laufzeit $O(kC)$ (Pseudo-Polynomialzeit); s-t-Schnitt (S, T) mit Kapazität $c(S, T)$, alle Kapazitäten beliebiger Schnitte sind obere Schranken für alle Flusswerte, MaxFlow/MinCut-Theorem, Korrektheit von `ford_fulkerson`, Polynomialzeit-Algorithmen für MAX FLOW.

13. Flussnetzwerke 2: Anwendungen

    13.1. Matchings in bipartiten Graphen
    Matching, perfektes Matching, größtes Matching, bipartiter Graph $G = (X \cup Y, E)$, MAX BIPARTITE MATCHING, Reduktion auf MAX FLOW, Korrektheit der Konstruktion, Satz: `ford_fulkerson` löst MAX BIPARTITE MATCHING in $O(mk)$.

    13.2. Ein Verteilungsproblem
    Verbraucher, Erzeuger, Bedarfswerte, Zirkulation, NETWORK CIRCULATION, Reduktion auf MAX FLOW, Korrektheit der Konstruktion, Satz: `ford_fulkerson` löst NETWORK CIRCULATION in $O(kD)$.

    13.3. Zirkulation mit unteren Schranken
    Untere Schranken für Mindestflusswerte, Zirkulation mit unteren Schranken, Reduktion auf NETWORK CIRCULATION.

14. Zusammenfassung, Fragerunde, Ausblick

Index

Literaturverzeichnis