Algorithmik

Sebastian Iwanowski
FH Wedel

10. Vorlesungswoche

Algorithmik 10

Matchings in Graphen (maximale Anzahl von Kanten)

Spezialfall: Matchings in bipartiten Graphen

Def.: Ein Fluss f heißt ganzzahlig $\Leftrightarrow$ f(u,v) ist ganzzahlig für jede Kante (u,v)

Def.: Zu gegebenem bipartiten Graphen G = ((V,U),E) wird ein Netzwerk G‘ gebildet mit:
G‘ hat eine Quelle s mit je einer gerichteten Kante zu allen Ecken aus V mit Kapazität 1.
G‘ hat eine Senke t mit je einer gerichteten Kante von allen Ecken aus U mit Kapazität 1.
G‘ enthält für jede Kante von G eine gerichtete Kante von V nach U mit Kapazität 1.

Satz: G hat ein Matching M mit |M|=k $\Leftrightarrow$ G‘ hat einen ganzzahligen Fluss mit |f| = k

Satz: Gegeben ein Netzwerk mit ausschließlich ganzzahligen Kapazitäten:
i) Dann ist der Wert des maximalen Flusses ebenfalls ganzzahlig.
ii) Es existiert ein maximaler ganzzahliger Fluss.

Beweis: i) folgt aus dem Satz von Ford-Fulkerson (Vorlesung 8, Folie 6).
ii) haben wir in der Vorlesung nicht bewiesen

Korollar: Das maximale Matching in G entspricht dem maximalen Fluss in G‘

Referenzen zum Nacharbeiten und Vertiefen:

Alt, Kap. 4.6
Cormen, Kap. 26.3

Algorithmik 10

Matchings in Graphen (maximale Anzahl von Kanten)

Algorithmen für bipartite Matchings und ganzzahlige Flüsse

Satz: Ein maximales bipartites Matching kann mit Hilfe von des Flussalgorithmus von Edmonds-Karp in O(nm) gefunden werden.
(Anm.: bessere Abschätzung über Wert des maximalen Flusses)

Verbesserungen:

Hopcroft-Karp: $O(n^{0,5}m)$
Alt et al.: $O(n^{1,5}(m/\log n)^{0,5})$ (ist eine Verbesserung für dichtbesetzte Graphen)

Satz: In Einheitsnetzen (Netzwerke mit Kapazität 1 für jede Kante) braucht der Algorithmus von Dinic nur $n^{0,5}$ Iterationen. Die inneren Operationen betragen nicht O(nm) wie im allgemeinen Fall, sondern nur O(m). Damit benötigt der Algorithmus von Dinic in Einheitsnetzen Laufzeit $O(n^{0,5}m)$.

Korollar: Die Laufzeit von Hopcroft-Karp für bipartites Matching ist auch mit dem Algorithmus von Dinic zu erreichen.

Referenzen zum Nacharbeiten und Vertiefen:

Alt, Kap. 4.7
Cormen, Aufgabe 26-7
Turau Kap. 7 (vor allem Literaturhinweise 7.6)

Algorithmik 10

Matchings in Graphen (maximale Anzahl von Kanten)

Techniken für Matchings in allgemeinen Graphen

Def.: Ein Erweiterungsweg ist ein Weg startend an einer freien Ecke, der abwechselnd freie Kanten und gematchte Kanten benutzt bis zu einer anderen freien Ecke

Def.: Eine äußere Ecke ist eine Ecke, die eine ungerade Nummerierung auf dem Erweiterungsweg hat, also die Startecke, die 3., 5., ... Sie befindet sich immer am Ende einer gematchten Kante (außer der Startecke).

Def.: Eine Blossom ist ein ungerader Kreis, der maximal gematched ist: Die Blossom hat 2k+1 Kanten, von denen genau k gematcht sind.

(in Vorlesung: Beispiel aus Hugh)

Anm. Eine Blossom wird bei der Suche nach Erweiterungswegen dadurch entdeckt, dass zwei äußere Ecken benachbart sind.

Referenzen zum Nacharbeiten und Vertiefen:

Laszlo Lovasz / Michael Plummer: *Matching Theory*, North Holland 1986, ISBN 9630541688, Kap. 9.1

James McHugh: *Algorithmic Graph Theory*, Prentice Hall 1990, ISBN 0130236152, Kap. 8.3

Christos Papadimitriou / Kenneth Steiglitz: *Combinatorial Optimization*, Dover 1998, ISBN 0486402584, Kap. 10

Algorithmik 10

Matchings in Graphen (maximale Anzahl von Kanten)

Algorithmus von Edmonds für allgemeine Graphen

Skizze:

1) Suche Erweiterungsweg:
   1a) Suche und starte mit ungematchter Ecke und leerem Weg EW.
   1b) Betrachte Nachbarn:
   Falls einer ungematched ist -> EW gefunden.
   Sonst erweitere EW um Kante zu Nachbarn und dessen gematchter Kante
   Falls eine Blossom gefunden wird, kontrahiere diese
   und fahre im kontrahierten Graphen fort.
2) Falls kein EW gefunden wurde -> Matching ist maximal.
   Sonst:
   2a) Dekontrahiere Graphen um alle zuvor gefundenen Blossoms
   2b) Ergänze EW auf ursprünglichem Graphen
   2c) Erhöhe Matching um eine Kante auf dem EW und fahre fort bei 1)

Referenzen zum Nacharbeiten und Vertiefen:

Laszlo Lovasz / Michael Plummer: *Matching Theory*, North Holland 1986, ISBN 9630541688, Kap. 9.1

James McHugh: *Algorithmic Graph Theory*, Prentice Hall 1990, ISBN 0130236152, Kap. 8.3

Christos Papadimitriou / Kenneth Steiglitz: *Combinatorial Optimization*, Dover 1998, ISBN 0486402584, Kap. 10

Algorithmik 10

Matchings in Graphen (maximale Anzahl von Kanten)

Algorithmus von Edmonds für allgemeine Graphen

Laufzeit: 1) Mit Details: $O(n^2)$ (nichttrivial, wurde in Vorlesung nicht bewiesen)

2) O(n) (klar)

Die äußere Schleife wird O(n) mal durchlaufen, weil sich jedesmal das Matching um eine Kante erhöht. -> Gesamtlaufzeit: $O(n^3)$

Korrektheit:

Satz 1: Matching ist maximal ⇔ Es gibt keinen Erweiterungsweg

Beweis: Hausaufgabe ! (siehe Papadimitriou)

Satz 2: Sei M ein Matching in G. G habe eine Blossom. G‘ sei der kontrahierte Graph
G hat Erweiterungsweg bzgl. M ⇔ G‘ hat Erweiterungsweg bzgl. M
(wurde in Vorlesung nicht bewiesen)

Referenzen zum Nacharbeiten und Vertiefen:

Laszlo Lovasz / Michael Plummer: *Matching Theory*, North Holland 1986, ISBN 9630541688, Kap. 9.1

James McHugh: *Algorithmic Graph Theory*, Prentice Hall 1990, ISBN 0130236152, Kap. 8.3

Christos Papadimitriou / Kenneth Steiglitz: *Combinatorial Optimization*, Dover 1998, ISBN 0486402584, Kap. 10

Algorithmik 10

String Matching

Aufgabe: Gegeben ein Text T mit n Zeichen und ein Muster P mit m Zeichen:
Finde die Startpositionen, an denen P in T vorkommt.

naiver Algorithmus: läuft in O(nm) Zeit

Algorithmus von Knuth-Morris-Pratt: läuft in O(n) Zeit

Def.: P_q, bezeichnet das Präfix von P, das aus den ersten q Zeichen besteht.

Def.: Die Präfixfunktion $\pi: \mathbb{N} \rightarrow \mathbb{N}$ für das Muster P ist definiert als:
$\pi(q) = k \Leftrightarrow$ k ist die Länge des längsten echten Präfixes von P_q, das gleichzeitig auch Suffix von P_q ist.

Prinzip des Algorithmus:

Zunächst wird für jedes $q \leq m$ die Präfixfunktion ausgerechnet und abgespeichert.
Dann wird der Text T in einem Durchlauf gescannt und das Muster P bei einem Mismatch entsprechend der Präfixfunktion verschoben.

Referenzen zum Nacharbeiten und Vertiefen:

Alt, Kap. 4.8
Cormen, Kap. 29.4

Algorithmik 10

String Matching

Algorithmus von Knuth-Morris-Pratt: läuft in O(n) Zeit

Implementierung des Hauptalgorithmus:

```
i := 1; q := 1;
while i ≤ n do
{
    if T[i] = P[q]
       then i := i+1
       else q := π (q-1);
    q := q+1;
    if q > m
       then
       {
            print („Treffer bei Position “, i-m);
            q := π (q-1);
       }
}
```

Hausaufgabe:
Beweisen Sie die lineare Laufzeit dieses Algorithmus unter der Voraussetzung, dass die Präfixfunktion schon ausgerechnet worden ist.

Referenzen zum Nacharbeiten und Vertiefen:

Alt, Kap. 4.8
Cormen, Kap. 29.4

Algorithmik 10

String Matching

Algorithmus von Knuth-Morris-Pratt: läuft in O(n) Zeit

Implementierung der Präfixfunktion (nach Cormen/Alt): läuft in O(m) Zeit

```
Π(0) := 0;
consequentMatch := 0;
for q := 2 to m do
{
    while (P(consequentMatch+1)≠P(q)) and (consequentMatch>0) do
        consequentMatch := π(consequentMatch);
    if P(consequentMatch+1)=P(q)
        then consequentMatch := consequentMatch+1;
    π(q) := consequentMatch
}
```

Hausaufgabe:
Beweisen Sie die lineare Laufzeit dieser Implementierung der Präfixfunktion

Referenzen zum Nacharbeiten und Vertiefen:

Alt, Kap. 4.8
Cormen, Kap. 29.4