Übungen zur Algorithmischen Bioinformatik I

Hausarbeit 1

Xiheng He

Mai 2021

2. Aufgabe: Analyse von SMSS

Naive Lösung:

Analyse der Laufzeit: In meiner Lösung verwendet der Algorithmus genau drei For Schleifen, Vergleichen Operationen sind >, == und >=. Die dritte Schleife erfordert eine Array-Addition und die zweite Schleife erfordert drei Vergleichen und fünf andere Operationen deshalb wird die Aufwand dieses Algorithmus nach Skript Seite 47 wie folgt:

$$A_{Naiv}(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=i}^{j} 1 = \frac{n^3 + 3n^2 + 2n}{6}$$

Damit wird $A_{Naiv}(n) \in O(n^3)$ bleiben

Analyse des Speicherbedarf: In diesem Algorithmus werden nur 7 Integer Variablen gebraucht damit beträgt der Speicherbedarf insgesamt $O(1)$

DP Lösung:

Analyse der Laufzeit: Der DP Algorithmus braucht genau zwei For Schleifen und in zweiter Schleife vier Vergleichen Operationen. Die Aufwand ist damit:

$$A_{DP} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \frac{n^2 - n}{2}$$

Deshalb ist die Komplexität der Laufzeit $O(n^2)$

Analyse des Speicherbedarf: Jede Variable wird als Integer gespeichert und Speicherbedarf ist $O(1)$

Divide & Conquer Lösung:

Analyse der Laufzeit: Die Eingabe wird in D&C Algorithmus halbiert und jeder Schritt benötigt noch genau eine Addition. Für eine Eingabe mit n benötigt damit $n - 1$ Additionen. Somit ist die Laufzeit:

$$A_{DC}(n) = A_{DC}(\lfloor (n/2) \rfloor) + A_{DC}(\lfloor (n/2) \rfloor) + (n-1) = n \log(n) - n + 1$$

Komplexität der Laufzeit ist daher $O(n \log(n))$

Analyse des Speicherbedarf: Der D&C Algorithmus benötigt Integer Variablen und genau eine 3 länge Int-Array um

Rückgabewerte zu speichern. Bedarf bleibt deshalb immer Konstant und die Komplexität beträgt $O(n)$

Lineare Lösung:

Analyse der Laufzeit: In Linearer Lösung ist nur eine For schleife vorhanden. Vergleichen Operationen sind $>$ und $=$. Alle Operationen sind Konstant und verändern sich nicht mit der Eingabe.

$$A_{linear}(n) \leq n \in O(n)$$

Analyse des Speicherbedarf: Der Algorithmus benötigt nur int Variablen daher ist Speicherbedarf Konstant $\in O(1)$

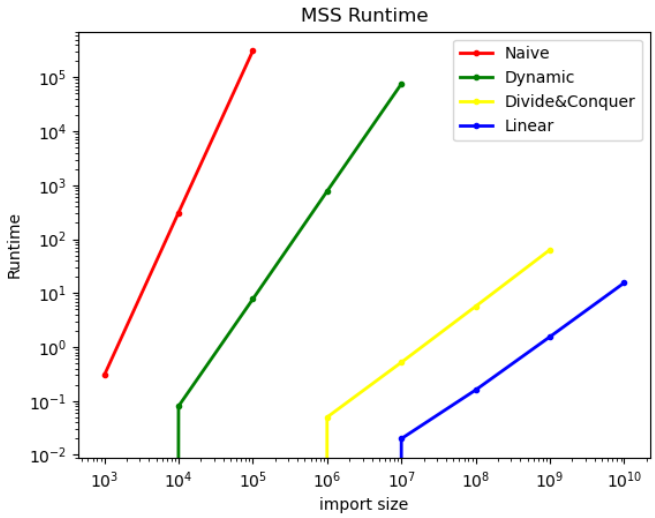

Abbildung 1: Laufzeit verschiedener SMSS-Algorithmen

4. Aufgabe: Analyse der Varianten

aSMSS:

Naive Lösung: Die erweiterte Naive Lösung erfordert detailliertere Vergleiche der Variablen in der dritten For-Schleife und speichert die vorübergehend erhaltenen Ergebnisse in einer Liste. Diese Operationen sind jedoch Konstanten und werden nicht erhöht, wenn sich die Eingabe ändert. Die Komplexität der Laufzeit wird damit nicht erhöht wenn sich die neue Operationen in dritter For-Schleife nicht mit der Eingabegröße verändern. Die Laufzeit-Komplexität der aSMSS Variante für Naive Lösung ist somit auch $\in O(n^3)$ Die Variablen der Naive Lösung sind int Variablen und ArrayList. Nicht konstant ist nur ein ArrayList für Speichern der alle SMSS. Offensichtlich ist: $||ArrayList|| \leq ||Eingabe||$, daraus ist worst-case Speicherbedarf $\in O(n)$ und best-case $\in O(1)$.

DP Lösung: wie oben erklärt sind die Veränderung des erweiterten Algorithmus nur detailliertere Vergleiche die immer Konstanten bleiben. Die Laufzeit bleibt ebenso $O(n^2)$. Nicht konstante Variable ist eine ArrayList und $\leq ||Eingabe||$. daraus ist worst-case Speicherbedarf $\in O(n)$ und best-case $\in O(1)$.

Divide & Conquer Lösung: Die Laufzeit beträgt $O(n \log(n))$ weil es nur Operationen mit konstanter Komplexität

hinzugefügt werden. Speicherbedarf ist ebenfalls worst-case $O(n)$ und best-case $O(1)$.

Lineare Lösung: Die Laufzeit beträgt $O(n)$ weil es nur Operationen mit konstanter Komplexität hinzugefügt werden. Speicherbedarf ist ebenfalls worst-case $O(n)$ und best-case $O(1)$.

LMSS:

Naive Lösung: Gesucht ist längest MSS in daher benötigt einen Vergleich für MSS länge jedoch hat der Vergleich nur $O(1)$ Komplexität. Die Laufzeit ist $O(n^3)$ wie in SMSS. In LMSS Algorithmus sind nur int Variablen benötigt, Speicherbedarf ist $O(1)$.

DP Lösung: Die Laufzeit ist ebenso $O(n^2)$ wie in SMSS. Speicherbedarf ist ebenso $O(1)$ wie in SMSS.

Divide & Conquer Lösung: Wie in SMSS ist Laufzeit $O(n \log(n))$ und Speicherbedarf $O(1)$.

Lineare Lösung: Geändert wurde nur Vergleichsoperation ≥ 0 statt > 0 damit werden nur LMSS gesucht. Wie in SMSS ist Laufzeit $O(n)$ und Speicherbedarf $O(1)$.

aLMSS:

Um alle LMSS zu finden benötigen alle Algorithmen genau eine ArrayList für Speichern deshalb sind die Speicherbedarfe aller Algorithmen in worst-case $O(n)$ und best-case $O(1)$.

Naive Lösung: Die Laufzeit beträgt wie in aSMSS $O(n^3)$.

DP Lösung: Die Laufzeit beträgt wie in aSMSS $O(n^2)$.

Divide & Conquer Lösung: Die Laufzeit beträgt wie in aSMSS $O(n \log(n))$.

Lineare Lösung: Die Laufzeit beträgt wie in aSMSS $O(n)$.

aMSS: Die Naive, DP und Divide & Conquer Lösung weiterhin genau eine ArrayList deshalb sind die Speicherbedarfe aller Algorithmen in worst-case $O(n)$ und best-case $O(1)$. Die lineare Lösung benötigt eine zusätzliche ArrayList, um die alle in Sequence gefundenen Teilfolgen zu speichern, deren Summen gleich 0 sind. Speicherbedarf für diese zusätzliche ArrayList ist in worst-case $O(n)$ und best-case $O(1)$. Der Speicherbedarf beträgt insgesamt ebenfalls worst-case $O(n)$ und best-case $O(1)$.

Naive Lösung: Die Laufzeit beträgt wie in aLMSS $O(n^3)$.

DP Lösung: Die Laufzeit beträgt wie in aLMSS $O(n^2)$.

Divide & Conquer Lösung: Die Anzahl der LMSS im MSS-Problem darf theoretisch $n/2$ nicht überschreiten. Um LMSS zu verkürzen braucht man in Divide & Conquer Lösung in worst-case $O(n^2)$ zusätzliche Operationen und in best-case $O(n)$. Die Laufzeit ist damit in worst-case $O(n^3 \log(n))$ und best-case $O(n^2 \log(n))$.

Lineare Lösung: In dieser Variante wird eine zusätzliche For-Schleife benötigt. Die Komplexität dazu kann zwar erhöht werden aber nicht größe als Eingabegröße n. In

worst-case beträgt $n-1$ mal pro Schleife, best-case 0 mal pro Schleife. Daher ist Laufzeit in worst-case $O(n^2)$, best-case $O(n)$.

5. Aufgabe: Anwendung von SMSS und aSMSS

a) SMSS in Yeast chromosome (header: »tpg|BK006935.2| [organism=Saccharomyces cerevisiae S288c])

startposition: 641 endposition: 229657 maxsocre: 28795 (sequence beginnt am 1)

b) Am sinnvollsten sollten nur Proteinsequenzen mit ca. 20 Aminosäure und maximal score , die auch eine starken Hydrophobizität darstellen, ausgegeben werden. Daher würde ich SMSS Algorithmus verwenden. Damit kann man auch die überlappende Teilfolgen ausschließen.

Um Transmembranproteine vorherzusagen solltet aSMSS eingesetzt werden. Für Proteom mit extrem größe kann lineare Lösung hier verwendet werden. c) Zu dieser Aufgabe habe ich einen Website erstellt.

Website

Hydrophobizitätsindex kann festgelegt werden.

Die Transformation muss jeweils neu durchgeführt werden. Die int Arrays können vorberechnet werden.