تحدّثنا في المقالِ السّابق (https://www.syr-res.com/article/14183.html) عن استخدامِ المؤتمتات في عمليّةِ تسريعِ البحثِ عن سلسلةٍ معيّنةٍ ضمنَ سلسلةٍ أكبرَ أو ضمنَ الجينومِ كاملاً. نتناولُ اليومَ بنيةً هامّةً في تمثيلِ البياناتِ وهي البُنى الشّجريّة.

ما المقصود بالبنيّةِ الشّجريّةِ Tree؟
البنيّةُ الشّجريّةُ هي إحدى طرقِ تمثيلِ البياناتِ والّتي كما يوحي اسمُها تتكوّنُ من جذرٍ ويتفرّعُ عنهُ العديدُ من الفروعِ، فهي مجموعةٌ من العُقدِ Nodes وتربطُ بينها أضلاعٌ Edges. تُدعى إحدى العقدِ بالجذرِ Root وكلُّ عقدةٍ عداها ترتبطُ بضلعٍ موجّهٍ إليها من عُقدةٍ ثانيّةٍ ليسَ أكثر. نلاحظُ في الشّكلِ التّالي بنيةً شجريّةً الجذرُ فيها هو العُقدة A وَكلُّ ضلعٍ فيها موجّهٌ من عقدةٍ أولى تُدعى العُقدةُ الأب إلى عقدةٍ أخرى تُدعى العقدةُ الابن. كمثال: العُقدة C تمثّلُ عقدةَ أب وكلٌّ من D وَ E وF هي عُقَدُ أبناء.

[[[[img:28937]]]]

كيف تُفيدنا هذهِ البُنى في عمليّةِ البحث؟
لنفترض أنّنا نبحثُ عن الكلمةِ pen ضمنَ النّصِّ التّالي happen فإنّنا فعليّاً يمكن أن نبدأ من نهايةِ النّصِّ ونقارن pen معَ كلٍّ من:
n، en، pen، ppen، appen، happen

وسنكتشفُ وجودَها عندَ تطابقها مع pen!
تُدعى هذه الأجزاءُ الّتي قمنا بتجزئةِ النّصِّ إليها "لواحق هذا النص Suffixes". هنا يأتي دورُ البنيّةِ الشّجريّةِ في تمثيلِ هذه اللّواحق وتسهيلِ البحثِ فيها. قبلَ أن نقومَ بذلكَ سنذكرُ ملاحظتَين هامّتَين:
1- عندَ تمثيلِ اللّواحقِ في البنيّةِ الشّجريّةِ سوفَ نضعُ الأحرفَ على الأضلاعِ وليسَ على العُقدِ.
2- حتّى نُميّزَ نهايةَ النّصِّ سوفَ نضعُ رمزاً يدلّنا على النّهايةِ وليكن $ وبالتّالي أصبحَ النّصُّ: happen$ والكلمة أصبحت pen$.
لاحظِ الشّكلَ التّالي:

[[[[img:28938]]]]

تُمثِّلُ هذه البنيّةُ الشَّجريّةُ كافةَ اللّواحقِ المُمكنةِ للنَّصِّ happen انطلاقاً من الجذر. والآن يمكنُ أن نبحثَ في هذه اللواحقِ عن الكلمةِ المطلوبةِ Pen ببساطةٍ عبر الانطلاقِ من الجذر واختيار الفرعِ المناسبِ وفقاً للحرفِ التّالي من الكلمةِ.
تبدأ الكلمةُ بالحرفِ p إذاً نختارُ الفرعَ الّذي يحملُ الحرفَ p، هنا لدينا خياران للمتابعةِ إمّا معَ الحرفِ p أو معَ الحرفِ e فنختارُ الفرعَ الّذي يحملُ الحرفَ e ونتابع لنحصلَ على فرعٍ يحوي كاملَ الكلمةِ pen، وهكذا وجدنا الكلمةَ المطلوبةَ ضمنَ النَّصِّ بطريقةٍ فعّالةٍ!
لو فرضنا أنّنا نبحثُ عن كلمةِ pet عندها سوفَ ننطلِقُ من الجذر ونختار الفرعَ الّذي يحملُ الحرفَ p ثمَّ الحرفَ e وعندها لن نستطيعَ المتابعةَ لعدمِ وجودِ فرعٍ يحملُ الحرفَ t، فالكلمةُ pet ليست موجودةٌ ضمنَ النَّصِّ happen.

[[[[img:28939]]]]

[[[[img:28940]]]]

كيف نعرفُ كم مرَّةً تكرَّرتِ الكلِمةُ في النَّصِّ؟
بحسبِ عددِ فروعِ الشَّجرةِ الّتي تصدرُ بعدَ انتهاءِ الكلمةِ ! في حالةِ البحثِ عن $pen لم يكن يوجد سوى فرعٌ واحدٌ للشّجرةِ وهو الّذي تابعنا الكلمةَ عليهِ، بالتّالي الكلمةُ مكرَّرةٌ مرّةً واحدةً. في المثالِ التّالي نبحثُ عن الكلمةِ an في النَّصِّ banana وَكما نلاحظ تتكرّرُ الكلمةُ مرّتين في النَّصِّ. إذا قمنا بتمثيلِ البحثِ وفقَ شجرةِ اللّواحقِ سوفَ نلاحظُ وجودَ فرعين للشّجرةِ بعدَ انتهاءِ الكلّمةِ ما يدلُّ على وجودِها مرّتين في النَّصِّ:

[[[[img:28941]]]]

لقد قمنا ببناءِ شجرةِ اللّواحقِ دونَ مراعاةِ التّرتيبِ الأبجديِّ للّواحقِ، فبدأنا من اليسار إلى اليمين باللّواحقِ الّتي تبدأ بالحرفِ n ثمَّ الّتي تبدأ بالحرفِ a ثمَّ الّتي تبدأ بالحرفِ b. عندما نقومُ بترتيب هذه اللّواحقَ وفقاً للتّرتيبِ الأبجديِّ، يمكن عندها أن نعطي لكلِّ لاحقةٍ رقماً يعبِّرُ عن ترتيبها بينَ اللّواحقِ بحيثُ تأخذُ أطولَ (لاحقةٍ وهيَ النَّصُّ ذاتُه) رقماً يعادلُ طولَ النَّصِّ ثمَّ ينقصُ العدد واحداً لأجل كلِّ لاحقةٍ إلى أن يأخذَ الرَّمزُ $ الّذي يوضعُ في نهايةِ النَّصِّ الرَّقمَ 0. كما يلي:

[[[[img:28942]]]]

يمكنُ الآنَ أن نقومَ بوضعِ هذهِ الأرقامَ بالتَّرتيبِ من اليسار إلى اليمين ضمنَ مصفوفةٍ تُدعى مصفوفةُ اللّواحقِ Array Suffix يُرمزُ لها بـ pos:
pos=[0،1،3،5،6،2،4]
الفائدةُ من هذه المصفوفةِ تكمنُ في سهولةِ وسرعةِ التّعاملِ معها أكثرَ من البنيةِ الشّجريّةِ. كيف يتمُّ ذلكَ؟
سنقومُ بشرحِ الطّريقةِ في المثالِ التّالي:
لدينا النَّصُّ ممثّلاً بـ $banana والكلمةُ الّتي نريد البحثَ عنها هي nan.
نكتب مصفوفةَ اللّواحقِ ونكتب ترتيبَ العناصر فيها:

pos=[0،1،3،5،6،2،4]
0 1 2 3 4 5 6
هدفنا إيجادُ المجالِ من هذه المصفوفةِ الّذي تقعُ فيهِ الكلمةُ المطلوبةُ. نحنُ نعلمُ أنَّهُ بترتيبِ اللّواحقِ أبجديّاً فإنّ هذهِ الكلمةَ سوفَ تقعُ في المرتبة 6:
$ (0) -> a$ (1) -> ana$ (2) -> anana$ (3) -> banana$ (4) -> na$ (5) -> nan$ (6)
إذاً يجبُ تحديدُ الحدِّ الأيمن والأيسر لهذا المجالِ.
نبدأُ بتعيين الحدِّ الأيمن:
نقومُ بتعريفِ مُتغيّرين هُما L=0 وَ R=6 حيثُ 6 طولُ النَّصِّ. ثمَّ نقومُ بحسابِ المُتوسطِ: M=(0+6)/2=3

إنّ اللّاحقةَ ذاتُ التّرتيبِ 3 في المصفوفةِ هي اللّاحقةُ ذاتُ الرَّقم 5 وَهي $anana (تذكَّر كيفَ قُمنا بمنح الأرقامِ للّواحق)
وَبمقارَنَةِ nan معَ $anana أبجديّاً نجدُ أنَّ nan تليها أبجديّاً، لذلكَ يُصبحُ المُتغيِّرُ L الجديدُ هو M ويبقى R على حاله.
نتابع مع L=3 وَ R=6 ثمَّ نقومُ بحسابِ المتوسطِ من جديدٍ: M=(6+3)/2=4.5 نأخذُ الرَّقْمَ الصَّحيحَ الأصغرَ وهو 4.
إنّ اللّاحقةَ ذاتُ التَّرتيبِ 4 في المصفوفةِ هي اللّاحقةُ ذاتُ الرَّقم 6 وَهي $banana.

و بمقارنةِ nan معَ $banana أبجديّاً نجدُ أنَّ nan تليها أبجديّاً، لذلكَ يُصبحُ المتغيِّرُ L الجديدُ هو M وَيبقى R على حاله.
نتابع مع L=4 وَ R=6 ثمَّ نقومُ بحسابِ المتوسِّطِ من جديدٍ: M=(6+4)/2=5
إنّ اللّاحقةَ ذاتُ التّرتيبِ 5 في المصفوفةِ هي اللّاحقةُ ذاتُ الرَّقْمِ 2 وَهي $na.
و بمقارنةِ nan معَ $banana أبجديّاً نجدُ أنَّ nan تليها أبجديّاً، لذلكَ يصبحُ المُتغيّرُ L الجديد هو M وَيبقى R على حالِهِ.
نتابعُ معَ L=5 وَ R=6 وهنا نُلاحظُ أنَّ الفرقَ بينَ القيمَتَينِ أصبحَ 1 لذلكَ نتوقّفُ هنا وَنأخذُ القيمةَ R لتكونَ الحدُّ الأيمنُ لمجالِ وجودِ nan في المصفوفة، وفعلاً فإنَّ اللّاحقةَ ذاتُ التَّرتيبِ 6 في المصفوفةِ هي اللّاحقةُ ذاتُ الرَّقْمِ 4 وَهي $nan.
بعمليةٍ مُشابهةٍ يمكنُ إيجادُ الحدِّ الأيسر وهوَ 6 ذاتُها لأنَّ الكلمةَ لم تتكرَّر سوى مرَّةً واحدةً في المصفوفة.

تُدعى هذه العمليّةُ بالبحثِ الثُّنائيِّ وهي تُساعدُ في البحثِ بسرعةٍ وكفاءةٍ.

مِنَ السَّهلِ كما رأينا أن نقومَ ببناءِ شجرةِ اللَّواحقِ لنصٍّ صغيرٍ، لكن ماذا لو كانَ مؤلَّفاً من آلافِ أو ملايينِ الأحرُفِ؟
سوفَ يأخذُ هذا بالتّأكيدِ وقتاً طويلاً، لكن تمَّ تطويرُ العديدِ من الخوارزميّاتِ الّتي تُساعدُ في تقليلِ الزّمنِ اللّازمِ.

كانت هذه محطَّتُنا قبلَ الأخيرةِ في مجالِ مُطابقةِ النُّصوصِ وتعرفنَّا كيف يمكن استخدامُ البُنى الشَّجريّةِ في هذه المهمَّةِ لنتابعَ في المقالِ التّالي الحديثَ عن مُطابقةِ النُّصوصِ التَّقريبيّةِ ومطابقةِ نصوصٍ مُتعدِّدَةٍ معاً.

المصادر:
Algorithms for Sequence Analysis Lecture Notes- Saarland University
Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences، Navarro،G. and Raffinot،M.
http://www.cs.cmu.edu/~clo/www/CMU/DataStructures/Lessons/lesson4_1.htm

المساهمون في المقال :

 إعداد: Dania S. Humaidan

 تدقيق علمي: Bassel Zeno

 تدقيق لغوي: Maissaa Markabi


صوت: Ghandi Safar Saado


تعديل الصورة: Ramy Ali


نشر: Sandra Sukarieh