Паралелна програма за минимално скелетно дърво в граф

Курсов проект

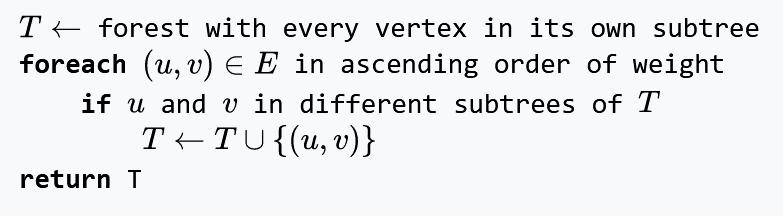
Васил Янакиев

Минимално скелетно дърво (на английски Minimum spanning tree – MST) е подмножество от ребрата на свързан неориентиран граф с тегло на ребрата, което свързва всички върхове заедно, без никакви цикли и с минималното възможно общо тегло на ребрата, т.е. това е дърво, чиято сума от теглата на ребрата е възможно най-малка. Съществуват много приложения на минималните простиращи се дървета. Един пример е телекомуникационна компания, която се опитва да положи кабел в нов квартал. Ако тя е ограничена да положи кабела само по определени пътища (напр. пътища), тогава ще има граф, съдържащ точките (напр. къщи), свързани с тези пътища. Някои от пътищата могат да бъдат по-скъпи, защото са по-дълги или изискват кабелът да бъде положен по-дълбоко; тези пътища ще бъдат представени с ребра с по-големи тегла. Валутата е приемлива единица за теглото на ребрата - няма изискване дължините на ребрата да се подчиняват на нормалните правила на геометрията, като например неравенството на триъгълника. Простиращото се дърво за този граф би било подмножество на тези пътища, което няма цикли, но все пак свързва всяка къща; може да има няколко възможни простиращи се дървета. Минималното дърво ще бъде това с най-ниски общи разходи, което представлява най-евтиния път за полагане на кабела.

Тъй като намирането на минимално скелетно дърво е широко разпространен проблем в теорията на графите, съществуват много последователни алгоритми за решаването му. Сред тях са алгоритмите на Крускал и Борувка, като всеки от тях използва различни свойства на MST. Всички те работят по сходен начин - подмножество на E се увеличава итеративно, докато се открие валиден MST. Тъй като обаче практическите проблеми често са доста големи (пътните мрежи понякога имат милиарди ребра), производителността е ключов фактор. Един от вариантите за подобряването ѝ е чрез паралелизиране на известните MST алгоритми.

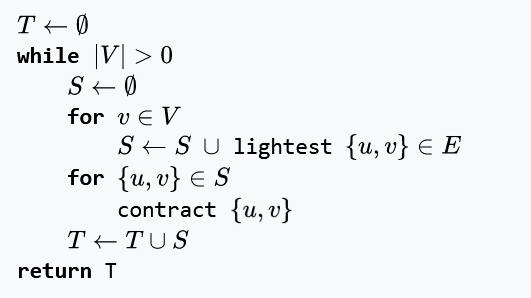
Ще разгледаме възможни паралелизации, и след това ще покажем имплементация на Борувка-паралелизацията.

Ще започнем с алгоритъма на Крускал. Алгоритъмът на Крускал използва свойството на циклите на MST. Псевдокод на високо ниво е представен по-долу.



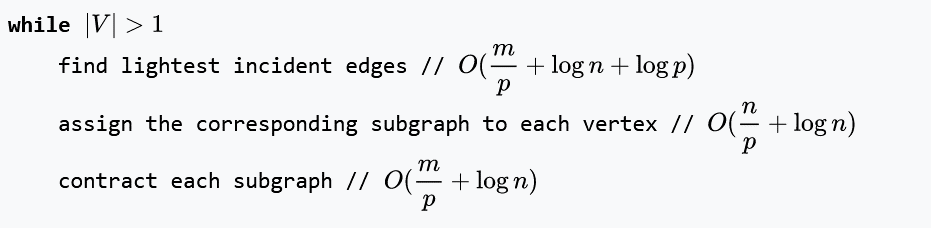
Поддърветата на T се съхраняват в структури от данни за намиране на съюзи, поради което проверката дали два върха са в едно и също поддърво е възможна за амортизирано време O ( α ( m , n ) ), където α ( m , n ) е обратната функция на Акерман. По този начин общото време за изпълнение на алгоритъма е в O ( с о р т ( n ) + α ( n ) ). Тук с α ( n ) се обозначава еднозначната обратна функция на Акерман, за която всеки реалистичен вход дава цяло число, по-малко от пет.

В метода на Крускал има части, които не могат да бъдат паралелизирани в класическия си вариант. Например определянето на това дали два върха са в едно и също поддърво е трудно за паралелизиране, тъй като две операции за обединяване могат да се опитат да присъединят едни и същи поддървета едновременно. Всъщност единствената възможност за паралелизация се крие в етапа на сортиране. Тъй като сортирането е линейно в оптималния случай на O ( log n ) процесори, общото време за изпълнение може да се намали до O ( m α ( n ) ).

Ще продължим с метода на Борувка. Основната идея на алгоритъма на Борувка е свиването на ребрата. Ребро { u , v } се свива, като първо се премахва v от графа и след това се пренасочва всяко ребро { w , v } ∈ E към { w , u } . Тези нови ребра запазват старите си тегла. Ако целта е не само да се определи теглото на даден MST, но и кои ребра включва, трябва да се отбележи между кои двойки върхове е свито дадено ребро. Псевдокод на високо ниво е представен по-долу.

Възможно е свиванията да доведат до множество ребра между двойка върхове. Интуитивният начин да се избере най-лекият от тях не е възможен в O ( m ). Въпреки това, ако всички свивания, които имат общ връх, се извършват паралелно, това е възможно. Рекурсията спира, когато остане само един връх, което означава, че алгоритъмът се нуждае от най-много log (n) итерации, което води до общо време за изпълнение в O ( m log (n) ).

Една възможна паралелизация на този алгоритъм дава полилогаритмична времева сложност, т.е. T ( m , n , p ) ⋅ p ∈ O ( m log n ) и съществува константа c, така че T ( m , n , p ) ∈ O ( log c m ). Тук T ( m , n , p ) означава времето за изпълнение на граф с m ребра, n върха на машина с p процесора. Основната идея е следната:

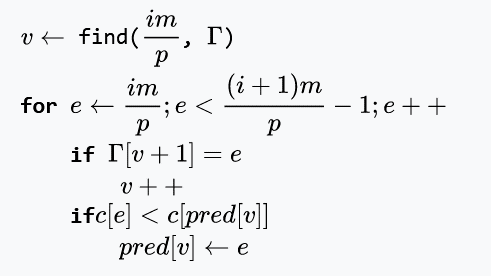


След това MST се състои от всички намерени най-леки ръбове.

При това паралелизиране се използва представянето на графа с масиви от съседство за G = ( V , E ). То се състои от три масива - Γ с дължина n + 1 за върховете, γ с дължина m за крайните точки на всеки от m ребра и c с дължина m за тежестите на ребрата. Сега за връх i другият край на всяко ребро, което е инцидентно за i, може да се намери в записите между γ [ Γ [ i - 1 ] ] и γ [ Γ [ i ] ]. Тежестта на i-тото ребро в Γ може да се намери в c [ i ]. Тогава i-тото ребро в γ е между върховете u и v тогава и само тогава, когато Γ [ u ] ≤ i < Γ [ u + 1 ] и γ [ i ] = v.

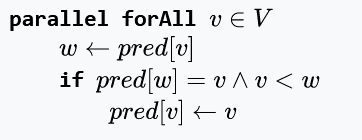
Най-напред ръбовете се разпределят между всеки от p-те процесора. i-тият процесор получава ръбовете, съхранени между γ [ im/p ] и γ [ ( i + 1 )m/p - 1 ] . Освен това всеки процесор трябва да знае към кой връх принадлежат тези ръбове (тъй като в γ се съхранява само една от крайните точки на ръба) и го записва в масива pred. Получаването на тази информация е възможно за O ( log n ), като се използват p двоични търсения, или за O ( n/p + p ), като се използва линейно търсене. На практика последният подход понякога е по-бърз, въпреки че е асимптотично по-лош.

Сега всеки процесор определя най-лекия ръб, падащ върху всеки от неговите върхове.

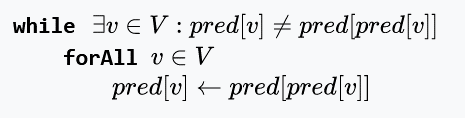


Тук възниква проблемът, че някои върхове се обработват от повече от един процесор. Възможно решение на този проблем е всеки процесор да има свой собствен масив prev, който впоследствие да се комбинира с масивите на останалите, като се използва редукция. Всеки процесор има най-много два върха, които се обработват и от други процесори, и всяка редукция е в O ( log p ). Така общото време за изпълнение на тази стъпка е O ( m/p + log n + log p ).

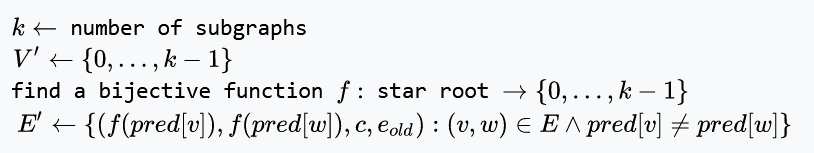
След това анализираме графа, който се състои само от ребрата, избрани в предишната стъпка. Тези ребра са насочени встрани от върха, към който са най-лекото падащо ребро. Полученият граф се разлага на множество слабо свързани компоненти. Целта на тази стъпка е да се присвои на всеки връх компонентата, от която той е част. Обърнете внимание, че всеки връх има точно едно изходящо ребро и следователно всеки компонент е псевдодърво - дърво с едно допълнително ребро, което върви успоредно на най-лекото ребро в компонента, но в обратна посока. Следващият код мутира това допълнително ребро в цикъл:



Сега всеки слабо свързан компонент е насочено дърво, в чийто корен има цикъл. Този корен се избира за представител на всеки компонент. Следващият код използва удвояване, за да присвои на всеки връх неговия представител:



В следващата стъпка всеки подграф се свива до един връх.

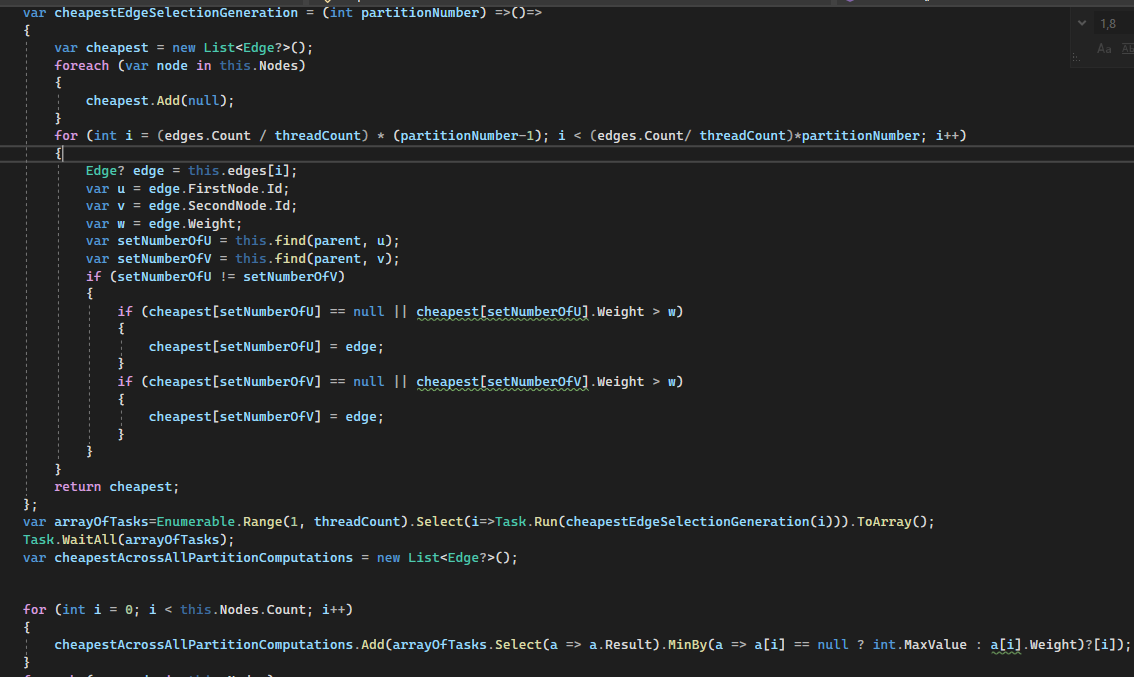


Намирането на биективната функция е възможно за O ( n/p + log p ), като се използва префиксна сума. Тъй като сега имаме нов набор от върхове и ръбове, масивът за съседство трябва да се възстанови, което може да се направи с помощта на Integersort върху E′ за O ( m/p + log p ) време.

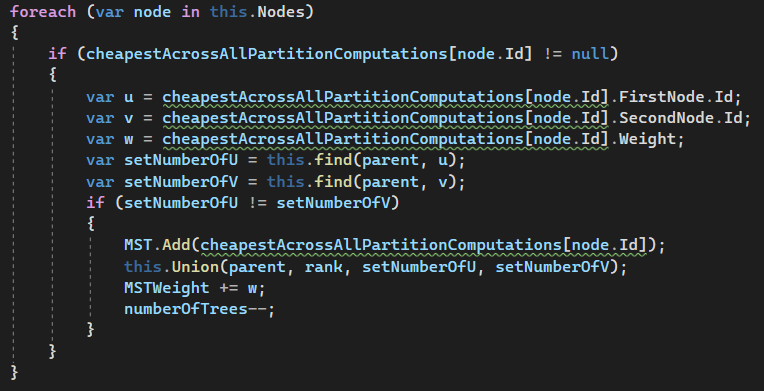
Всяка итерация сега се нуждае от O ( m/p + log n ) време и както в последователния случай има log n итерации, което води до общо време за изпълнение от O ( log n ( m/p + log n ) ). Ако m ∈ Ω ( p log^2 p ), ефективността на алгоритъма е в Θ ( 1 ) и той е сравнително ефективен. Ако m ∈ е O ( n ), тогава той е абсолютно ефективен.

Нашата имплементация на Борувка-паралелизацията има някои недостатъци. Начинът, по който се обработват данните – разпределянето на ребра между процесорите, а не на върхове – налага извършването на допълнителни операции за изчисляване на реброто с най-малко тегло. Също така успяхме да паралелизираме само тази, изчислителна стъпка – а изпълнението на присвояването на подграфите към отделни върхове остава изцяло последователно.

Това е изчислителната част:



Това е частта, свързана с присвояването на подграфите и свиването:



Въпреки това, програмата има сравнително добра производителност. Изчисляването на MST на граф с 5000 върха и 5000^2 ребра отнема около 1 секунда.

