

Задачи с дървета

Динамично програмиране в дърво

В едно динамично програмиране винаги говорим за това какъв е стейта. При динамичното в дърво стейтът е по-абстрактен, но най-общо казано, той по някакъв начин е свързан с поддървета. За пример ще дадем задачата за намиране на **диаметър в дърво**. Освен алгоритъма с пускането на DFS или BFS 2 пъти, има и друго решение, което използва дп.

Избираме който и да е връх за корен. Нека $dp[x]$ пази на какво разстояние е най-отдалечения връх в поддървото на x . Нека децата на x са v_1, \dots, v_k . Тогава имаме следната формула: $dp[x] = \max(dp[v_1], \dots, dp[v_k]) + 1$. След като сме сметнали дп-то за всеки връх, остава да пресметнем отговора. За всеки връх x търсим какъв е максималният път, който минава през върха x и е изцяло в поддървото на x . Ако x има точно едно дете, то това е просто $dp[x]$. Ако x има повече от 1 дете, то намираме кои са максималните две стойности измежду $dp[v_1], \dots, dp[v_k]$. Нека ги кръстим $max1$ и $max2$. Тогава това, което търсим, е $max1 + max2 + 2$. След като за всяко x сме сметнали търсената стойност, остава да вземем максималната.

Нека сега разгледаме тази задача: Трябва да се намери **максималния matching в дърво**. Коренуваме дървото във връх 1. За всеки връх има 2 възможности - може да изберем някое ребро, водещо до дете на x , или може да изберем реброто, водещо до родител на x . Затова ще имаме следните стейтове:

- $dp[x][0]$ който ще пази максималния мачинг за поддървото на x , където имаме право да избираме ребро, водещо до дете на x .
- $dp[x][1]$, който ще пази максималния мачинг за поддървото на x , където сме избрали реброто, водещо до родителя на x .

Отговорът ще бъде $dp[1][0]$, ако сме избрали 1 за корен. Сега да видим как да сметнем тези 2 стейта за конкретен връх. Нека v_1, \dots, v_k са деца на x .

Ако сме избрали реброто до родителя на x , то за v_1, \dots, v_k имаме възможност да изберем ребро, водещо до тяхно дете. Тоест $dp[x][1] = dp[v_1][0] + \dots + dp[v_k][0]$.

Ако имаме право на избор на ребро, водещо до дете, то трябва да изберем кое е най-доброто ребро. Нека сме избрали някое дете v_j . Тогава мачингът за поддървото на x ще бъде $dp[x][1] - dp[v_j][0] + dp[v_j][1]$ (само стейтът на v_j е 1, останалите са със стейт 0). $dp[v_j][1] - dp[v_j][0]$ е така да се каже стойността, която ще получим, ако променим стейта на v_j от 0 на 1. Тоест $dp[x][0] = dp[x][1] + \max(dp[v_j][0] + dp[v_j][1])$.

Нека сега разгледаме тази **задача**. Коренуваме дървото във връх 0. Тук стейтът е много подобен на този от предишната задачата:

- $dp[x][0]$ - максималното щастие за поддървото на x , ако няма камион, който да ходи от x до родителя си.
- $dp[x][1]$ - максималното щастие за поддървото на x , ако задължително изпратим камион от x до $par[x]$ (родителят на x), като щастиято на това ребро влиза в отговора. Ако няма камиони във връх x , то тогава считаме, че $dp[x][1] = -1$.

За всеки връх, който не е корен и който има поне един камион в себе си, за начална стойност ще сложим $dp[x][1] = \text{щастиято на реброто}(par[x], x)$. Виждаме, че пресмятането на $dp[x][0]$ и $dp[x][1]$ е много подобно, единствената разлика е, че за $dp[x][1]$ имаме право да прашаме към децата на x един камион по-малко. Затова ще се фокусираме само върху пресмятането на $dp[x][0]$, за $dp[x][1]$ е аналогично.

Нека разгледаме случая, в който няма камиони във връх x . Тогава лесно се вижда, че

$dp[x][0] = \sum_{1 \leq j \leq k} \max(dp[v_j][0], dp[v_j][1])$. Трябва да намерим оптималните p_x на брой деца, към които да пратим камионите. Подобно на предишната задача за фиксирано дете v_j имаме щастие, което ще получим, ако пратим камион към него. Това щастие е точно $dp[v_j][0] + w_j - \max(dp[v_j][0], dp[v_j][1])$, където w_j е теглото на реброто (x, v_j) . За всяко дете смятаме тази стойност и избираме най-големите p_x такива стойности.

Динамично програмиране в дърво - rerooting идеи

В някои задачи може да се наложи да смятаме за всеки връх динамичното, ако този връх е корен на дървото. Ще правим нещо по-умно от това n пъти да пускаме динамичното.

Ще демонстрираме идеята с една по-проста [задача](#). Това е същото динамично като това за диаметър. Но този път искаме да намерим колко е $dp[x]$, ако x е корен на дървото. Ще коренуваме дървото в някой връх (да речем във връх 1). Сега имаме 2 възможни стойности за динамичното във връх x :

- $dp[x][0]$ - Както преди, това ще бъде дължината на максималния път от връх x до някой връх в поддървото му.
- $dp[x][1]$ - Максималният път от x до който и да е връх, където пътят задължително минава през родителя на x .

Първата стъпка е да сметнем $dp[x][0]$ за всеки връх x , което вече знаем как да направим. След това как да сметнем $dp[x][1]$? Ще пресмятаме стойностите отгоре-надолу, започвайки от корена. Тоест, когато се намираме във връх x , ще смятаме $dp[v_1][1], \dots, dp[v_k][1]$, като считаме, че стойността на $dp[x][1]$ вече е пресметната (от корена няма път нагоре, така че $dp[x][1] = -\infty$).

Да се фокусираме върху връх v_j . Виждаме, че е в сила следната формула: $dp[v_j][1] = \max(dp[x][1], \max(dp[v_p][0] + 1; 1 \leq p \leq k, p \neq j)) + 1$. Но виждаме, че тя е твърде бавна за пресмятане, защото за връх x трябва да извършим k^2 операции. Трябва да можем константно да пресметнем $dp[v_j][1]$. За да сметнем колко е максимумът за всичките дп-та на децата, с изключение на конкретното дете v_j , е достатъчно да пазим двата максимума (да ги кръстим $max1$ и $max2$) измежду $dp[v_1][0], \dots, dp[v_k][0]$. За конкретно дете v_j имаме 2 случая:

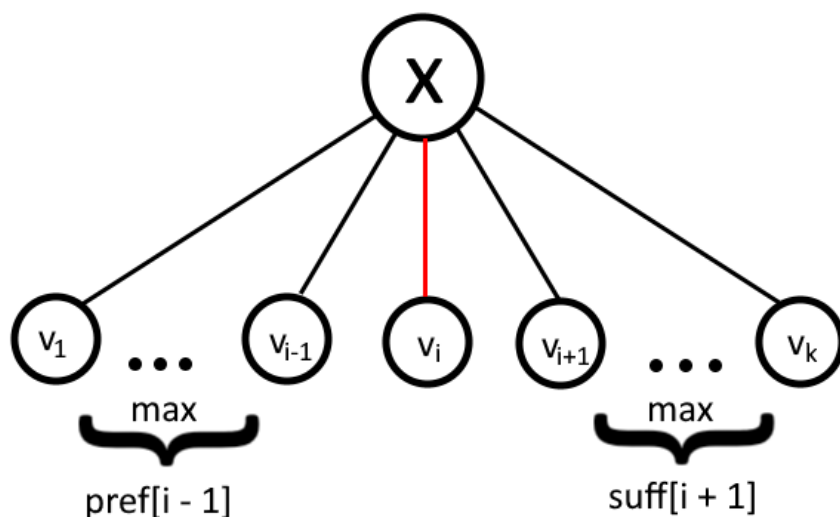
- $dp[v_j][0] = max1$ - тогава $dp[v_j][1] = \max(dp[x][1], max2 + 1) + 1$
- $dp[v_j][0] \neq max1$ - тогава $dp[v_j][1] = \max(dp[x][1], max1 + 1) + 1$

Отговорът за връх x е $\max(dp[x][0], dp[x][1])$.

Ключовото в такива задачи е как константно (или в някои екстремни случаи логаритмично) да сметнем специалното динамично, което се занимава с родителя. Сега ще покажем втори начин, който за тази задача е прекалено сложен, но за други задачи може да се окаже полезен.

Идеята е да използваме префиксни и суфиксни масиви (да ги кръстим $pref$ и $suff$). Минаваме през списъка на децата на x отляво надясно. Искаме $pref[j]$ да пази максималното $dp[v_p][0]$, където $1 \leq p \leq j$. Тоест $pref[j] = \max(pref[j-1], dp[v_j][0])$. Аналогично правим и за $suff$, като минаваме през списъка отдясно наляво.

Отново минаваме през списъка на децата на x . За детето v_i имаме:
 $dp[v_i][1] = \max(dp[x][1], \max(pref[i-1], suff[i+1]) + 1) + 1$.



Този подход с префиксните и суфиксните масиви е доста по-универсален, тъй като в масивите могат да се пазят суми, произведения и др. Накрая на този файл има задачи, в които се ползва този подход.

Последната задача от тази тема, която ще разгледаме, е [тази](#). Нека фиксираме връх 1 да ни е корен на дървото и да видим какво става, когато Дени постави пионката там. Тогава трябва да разглеждаме само случая, в който пионката отива от връх до негово дете. Дефинираме стойта $dp[u][0]$:

- 0, ако играчът, който е наред, когато пионката е на връх u , губи, като има право да мести пионката само до някое от децата на u .
- 1, ако в тази ситуация печели.

Ако за всички деца v_1, \dots, v_k на x е изпълнено, че $dp[v_j][0] = 1$, то $dp[x][0] = 0$. В противен случай, ако има поне едно "губещо дете играчът ще иска да сложи пионката там, така че в този случай $dp[x][0] = 1$.

Както в предишната задача, ще направим още един стойт, който е за родителя. Тук е важна прецизната дефиниция на стойта: $dp[u][1]$ ще пази дали играчът, който е на ход, **когато пионката е на връх $par[u]$** , може да спечели, ако **няма право да мести пионката на връх u** . Естествено тази дефиниция не важи за корена.

Разглеждаме връх x и се опитваме да сметнем $dp[v_1][1], \dots, dp[v_k][1]$. Отново считаме, че сме сметнали $dp[x][1]$. Ако x не е корен и $dp[x][1] = 0$, то тогава играчът на връх x (все пак за дете v_j е изпълнено, че $par[v_j] = x$) ще иска да премести пионката нагоре към $par[x]$, защото това е губеща позиция за опонента. Тоест в този случай $dp[v_1][1] = \dots = dp[v_k][1] = 1$.

В другия случай x или е корен, или $dp[x][1] = 1$. Играчът на x вече не може да сложи пионката на $par[x]$. Когато искаме да сметнем $dp[v_j][1]$, се стремим да намерим губеща позиция, в която да пратим играча от връх x . Затова разглеждаме 3 случая:

- Измежду $dp[v_1][0], \dots, dp[v_k][0]$ има 2 губещи позиции. Това означава, че дори и от връх x да не можем да отидем до точно едно от децата, то винаги ще можем да пратим опонента в губеща позиция. Тоест $dp[v_1][1] = \dots = dp[v_k][1]$.
- Измежду $dp[v_1][0], \dots, dp[v_k][0]$ има точно 1 губеща позиция. Нека тази губеща позиция е при детето v_j . Когато не пресмятаме $dp[v_j][1]$, имаме възможността да пратим пионката в губеща за противника позиция. Тоест $dp[v_p][1] = 1$ за всяко $p \neq j$. Но $dp[v_j][1] = 0$, защото този стойт съответства на това да сме във връх x и да нямаме право да отидем до v_j . Тоест където и да пратим пионката, тя ще отиде в печеливша за противника позиция.

- Измежду $dp[v_1][0], \dots, dp[v_k][0]$ няма губещи позиции. Тогава което и дп да пресмятаме, няма как да пратим противника в губеща позиция. Тоест $dp[v_1][1] = \dots = dp[v_k][1] = 0$.

След като сме сметнали двете дп-та, остава да видим какво е условието Дени да спечели, ако сложи пионката на връх x . Тези 2 условия трябва да изпълнени едновременно, за да може Дени да спечели:

- $dp[x][0]$ трябва да бъде 0. Това е, за да подсигури, че когато Боби е на ход във връх x той няма как да спечели, движейки пионката към някое от децата на x .
- $dp[x][1]$ трябва да е 1. Помним, че $dp[x][1]$ според нашата дефиниция пази дали ще спечели този, който е на връх $par[x]$. Тоест, ако Боби реши да премести пионката на $par[x]$, трябва Дени да е в печеливша позиция в $par[x]$.

Допълнителни задачи

- [Tree Distances II](#)
- [Subtree](#)
- [BOI 2017 Citi Attractions](#)
- [APIO 2010 Patrol](#)
- [EJOI 2017 experience](#)
- [Codeforces 1249F](#)
- [APIO 2014 Beads](#)
- [Codeforces 1349D](#)

Автор: Велислав Гърков