Logic Programming

Section #9

Eman Safwat & Shaimaa

23/12/2009

Sorting Algorithms

1) Insertion Sort

في الأول نفتكر مع بعض ال insertion Sort هوا أني امسك element وأشوف مكانه الصحيح فين أو بمعنى آخر مكانه اللي هيديني ال Sorted ..Array في الآخر.

طيب نعمل الكلام ده إزاي بال prolog؟؟

احنا الأول هنعمل Predicate اسمها insert_sorted ودي وظيفتها انها ت insert.. element في list sorted وطبعا في مكانه الصحيح (وبعدين هنكمل عليها إن شاء الله).

ال Predicate دي بتاخد sorted list وال integer اللي عايزين نعمله insert وهيرجعلنا ال list الجديدة.

insert_sorted ([1, 3, 5] , 4, L).

L= [1,3,4,5]

في ال Predicate دي أنا بقوله check على كل element في list وطول مال head أصغر من X خُط ال head زي ماهوا وأول متلاقي element أكبر من X خُط X وبعدين كمل باقي ال list.

ولو كل ال elements اللي في ال list أصغر من ال integer خُط ال integer ده في آخر ال list.

Domains

ilist = integer*

Predicates

insert_sorted([], X, [X]).

Base step ودي هتحصل لما ال list تبقىempty قبل ما أحط ال elements يعني مثلا في المثال اللي فات لو كان أدانى 6 integer بدل 4 ساعتها ال list كانت هتبقى empty قبل ما insert ال element عشان كده هيحطه في آخر ال list.

insert_sorted([H|T], X, [H|T1]):-

X >= H, !,

insert_sorted(T, X, T1).

لما X تبقى أكبر من ال Head معنى كده إن ده مش مكانها فهتنده على نفسها تاني بال Tail.

insert_sorted(L, X, [X|L]).

لو ال X اصغر هأضيفها ك Head لل List بتاعتي دي وبس كده.

insert_sorted بتفترض إن ال List اللي هت Insert فيها جايلي sorted أصلا.. فلو حد ادها list مش sorted هت insert ال Element عادي جدا وملهاش دعوة بال list مش بتعملها Sorting.

طيب دلوقتي بقى هنشوف أستخدم insert_sorted إزاي عشان أعمل ال insertion sort كله.

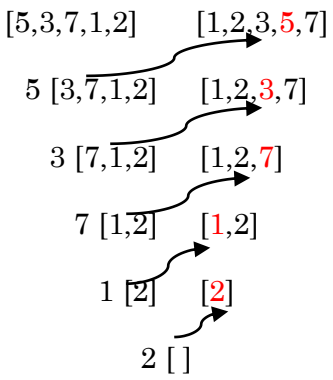
دلوقتي أنا عندي ال list دي [5,3,7,1,2] عشان اعملها insertion sort لازم امسك أول element واشوف مكانه الصحيح زي ماقلنا.

أنا هنا هاعمل sort لل Tailوبعدين أعمل insert لل Head في مكانها بإستخدام insert_sorted.

وبطريقة recursive هيوصل لل empty list وهتبقى ال head بتاعتها 2 وبعدين 1..

and so on..

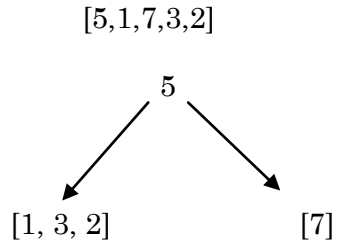
```
insertionSort([],[]).

insertionSort([H|T],L):-

insertionSort(T,ST),

insert_sorted(ST,H,L).
```



2) Quick Sort

مبدئيا هنعمل predicate هنسميها split_pivot وبتاخد list وrumber.. as a pivot وبناء على الرقم ده هيقسم ال list ل 2 lists.

واحدة من ال 2 Lists دول فيها Numbers أقل من ال Pivot والتانية أكبر.



Predicates

split_pivot(ilist,integer,ilist,ilist).

Split_pivot بتاخد list و ineteger اللي هتقسم بيه، وبترجع 2 Lists واحدة فيها ال Values الأكبر والتانية فيها الأصغر.

Clauses

split_pivot([],_,[],[]):-

Base Step أما تبقى ال empty List فيرجع ال 2 Lists.. empty برضو.

split_pivot([H|T],P, T1,[H|T2]):-

H>P,!,

split_pivot(T,P,T1, T2).

أما ال Head تبقى أكبر من ال Pivot انده على نفسك تاني بال Tail، ورجعلي ال Head مع T2 (ال List فيها ال Values الكبيرة).

split_pivot([H|T],P,[H|T1],T2):-

split_pivot(T,P,T1,T2).

هنا العكس أما تبقى ال Head أكبر رجعلي ال Head في List ال Values الصغيرة (T1).

Goal

split_pivot([4,3,7,2,0], 5, T1 ,T2).

T1=[4,3,2,0] T2 =[7]

طيب now بقى عايزين نعمل quick sort باستخدام split pivot

طيب نفتكر مع بعض ال quick sort برضو ..

ال quick sort كان بيقسم ال array ل 2 sub arrays ويعملهم sort هما الإتنين وبعدين يعملهم concatenation مع بعض.

طيب تمام كده يبقى هنستخدم predicate concatenate اللي احنا عملناها زمان..

نشوف بقى predicate .. quickSort:

Predicates

quickSort(ilist,ilist).

بتاخد list وترجعها sorted.

Clauses

quickSort([],[]).

quickSort([H|T],L):-

split_pivot(T,H,T1,T2),

هوا هنا اخد ال Head ك Pivot ل Split_pivot وهيقسم على اساسه ال List ل 2 Lists.

quickSort(T1,ST1),

quickSort(T2,ST2),

بياخد ال 2 Lists اللي رجعوله من Split وهيعمل Sorting لكل واحدة فيهم ويرجعهم Sorted.

conc(ST1,[H|ST2],L).

بعدين يعمل Concatenation لل 2 Sorted Lists وال Head في List كبيرة L ويرجع L وتبقى هيا دي الحل.

هوا بيحط ST1 (فيه ال Values الاقل) بعدين ال Head بعدين ال ST2 (فيها ال Values الأكبر) في L عشان تبقى مترتبة.

conc ([], L, L).

conc ([H | T], L, [H | TL]):-

conc (T, L, TL).

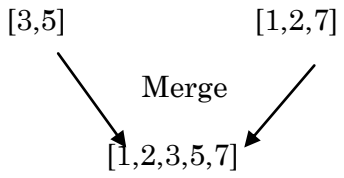
Goal

quickSort([4,3,7,2,0], L).

L=[0,2,3,4,7]

3) Merge Sort

في الاول احنا هنعمل predicate اسمها merge ودي هت merge 2 sorted lists بس لازم تطلع sorted list في الآخر.



يعني احنا في ال Merge Sort زي ماخدنا في ال Algo بنقسم ال List ل 2 Lists ونعملهم sorting كل واحدة لوحدها، وبعدين نعملهم merge بإننا كل مرة نقارن كل Element من أول List مع أول Element من تاني List والأصغر منهم هيتحط الأول وبعدين أقارن ال Element الأكبر بباقي ال List وكده مع كل ال Elements لحد مال 2 Lists يبقوا Empty.

احنا دلوقتي مش هنعمل ال Merge Sort كله.. هنعمل ال merge predicate اللي بتاخد 2 Lists بعد مايتعملهم Sorting وتقارنهم ببعض لحد ماتعمل List كبيرة Sorted. وبإستخدامها اعملوا انتوا بقية ال Merge Sort ك Assignment.

Predicates

merge (ilist,ilist,ilist).

بتاخد 2 Lists وترجع List كبيرة بال 2 Lists دول معمولها Sorting.

Clauses

merge([],L,L).

Base Step 1 لما أول List تبقى Empty ساعتها هيرجع ال List التانية بس.

merge(L,[],L).

Base Step 2 لما تاني List هيا اللي Empty وهيرجع أول List.

هما ليه 2 Base Steps؟ ليه مش واحدة بس زي كل مرة؟ عشان أنا مش ضامنة إنه ال 2 Lists يبقوا ليهم نفس ال Length، ممكن واحدة تكون أطول من التانية.. فكده هيبقى فيه واحدة تخلص بدري، فال 2 Base Steps عشان ت Handle كل ال Cases دي.

merge([H1|T1],[H2|T2],[H1|T]):-

H1<H2,!,

هنا بقوله لو Head1 من أول List أصغر من Head2 من تاني List، معني كده إن Head1 هوا ده مكانه في List ال Output (أصغر Element يعتبر في ال 2 Lists.. على أساس إنهم جايين Sorted).

merge(T1,[H2|T2],T).

بنده على نفسي تاني بال Tail لأول List وبال List التانية كلها عشان أنا كده اخدت Head1 بس وحطيطها في مكانها في ال Output.. Head2 لسة وترجع T.

merge([H1|T1],[H2|T2],[H2|T]):-

merge([H1|T1],T2,T).

تاني Recursive Step لو Head2 هوا الأصغر يبقى هوا اللي هيتضاف في الOutput.. والباقي نفس الكلام.

Goal

merge([3,5],[1,2,7],L).

L=[1,2,3,5,7]

طيب احنا دلوقتي عشان نعمل merge sort لازم اقسم list اللي هتجلي 2 list قد بعض

طيب دلوقتي بقى هنعمل predicate اللي هتقسملنا list وهنسميها split.

split(L,L1,L2):-

conc(L1,L2,L),

هنا conc بستعملها بالعكس.. بـأديها L معمولها Binded ترجعلي L1 و L2 لكل ال Possible Combinations بين L1 و L2.

len(L1,N1),

len(L2,N2),

len دي بترجع length ال List.

abs(N1-N2)<=1,!.

abs دي Built-in Predicate.. عشان يرجعلي 2 Lists ليهم نفس الطول لو ال List الكبيرة even ويرجعهملي واحدة أكبر من التانية بواحد عشان لو الList الكبيرة odd.

Assignment 1:

Use merge and split predicates to make "merge sort" predicate.

Binary Dictionary Trees

إيه هيا ال Binary Dictionary Tree؟

هيا Tree فيها كل node ليها 2 branches بس left و right. Left Node بتبقى أقل من ال Parent Node وال Right Node بتبقى أكبر من ال Parent Node.. من الآخر هيا ال BST (Binary Search Tree) اللي اخدناها في ال Data Structures بس غيرت اسمها p:

طبعا ال Prolog مفهاش Data Type مخصوص لل Trees فهنعملها إزاي؟ هنعمل Compound Object جواه Compound Object من نفس النوع (عشان كل Node يبقى جواها 2 Nodes زيها).

Domains

```
tree = t(tree, integer, tree) ; null
       ↙       ↓        ↘
Left Node   Root    Right Node
```

أنا كده عرفت Compound Object اسمه tree وجواه 3 Variables اتنين من نوع tree برضو و integer.

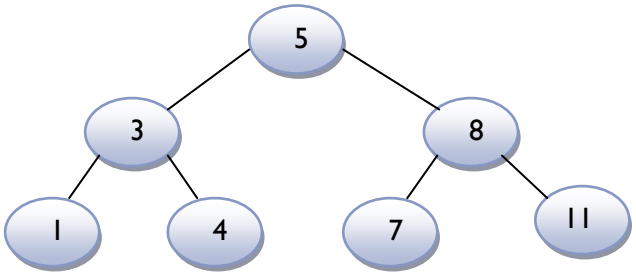
null ; دي معناها إنه ممكن ميكونش جواه أي حاجة من الحاجات دي ويبقى بـ null (هوا هنا مفهمش إيه دي ومعندوش حاجة Built-in اسمها null بس أنا اللي هفهمه أنا أقصد إيه بـ null دي جوه ال Code - وعامة أي رمز عايز تستعمله في ال Program بتاعك ممكن تعرفه في ال Domains وبعدين تستعمله عادي في ال Code).

T = t(t(t(null, 1, null), 3, t(null, 4, null)), 5, t(t(null, 7, null), 8, t(t(null, 9, null), 11, null)))

أنا كده عملت Representation لل Tree في One Variable اللي هوا T، عشان لما استعمل T بعد كده يبقى فاهم إنها بتعبر عن ال Tree كلها.

الأرقام دي بتمثل ال Root لكل tree واللي على يمينها وشمالها دول ال Right Node وال Left Node ليها، يعني مثلا ال 1 ال Right Node ليها بـ null وال Left Node كذلك وده معناه إنها Leaf.

ال 5 مثلا دي Root ل tree ال Right Node بتاعتها tree برضو وال Root لل tree دي بـ 8، نفس الكلام بالنسبة لل Left Node هيا tree وال Root ليها بـ 3.. لو مش فاهمين أوي أهيه شكل ال Tree:



كل Root اللي على شمالها اصغر منها واللي على يمينها أكبر منها، زي ماقلنا.

In Trees:

هنعمل Predicate اسمها in بتاخد tree و Integer وترجعلي Yes Or No إذا كان ال Integer ده موجود في ال tree ولا لأ (زي member اللي كنا اخدناها قبل كده).

Domains

tree = t(tree, integer, tree); null

عرفت Compound Object اسمه tree زي معملنا فوق.

Predicates

in(tree, integer).

بأديها tree و Integer وهيا ترجع Yes أو No.

Clauses

in(t(_, X, _), X):- !.

Base Step لو X (اللي أنا مديهاله) بتساوي ال Root يبقى خلاص موجودة، اخرج ورجع yes.

in(t(L, Y, _), X):-

X < Y, !,

in (L, X).

Recursive Step 1 لو X أقل من Y (ال Root) يبقى انده على نفسك بس بال Left Node (عشان ال Left دايما بيبقى فيه ال Values اللي أقل من ال Root) وال Cut اتحطت عشان لو X<Y اتحقق ميعملش Backtrack على in التالتة.

in(t(_, _, R), X):-

in(R, X).

طيب لو X كانت أكبر من Y، انده على نفسك بس المرادي بال Right Node عشان عكس اللي فات (كل ال Values فيها أكبر من ال Root).

طبعا كل ال Underscores دي عشان لو حطيت Variables بيدي Warnings إن هما Unused Variables.

Goal

in (t(t(t(null, 1, null), 3, t(null, 4, null)), 5, t(t(null, 7, null), 8, t(t(null, 9, null), 11, null))), 4).

اديتله ال Tree بتاعتي وال integer اللي هي Check عليه (4).. هترجع yes.

لو مديالله integer مش في ال Tree هيدخل في التلاتة in فميلاقيش ال X بت Match مع أي Root فهيخرج ويرجع no. اعملوا Tracing مع نفسكوا لل 4 وشوفوا هت Match إزاي.

Height of Trees:

هنعمل دلوقتي Predicate اسمها height بأديها tree وبترجعلي integer بال height بتاع ال tree دي، height يعني عدد ال Levels بتاعت ال tree من أول ال Parent Node فوق لحد ال Leaves تحت.

Predicates

height(tree, integer).

عرفت height بتاخد tree وترجعلي integer.

max(integer, integer, integer).

max بتاخد 2 integers وترجع ineteger ال maximum بين الإتنين.

Clauses

height(null, 0).

Base Step ل height أما نوصل لل Leaves ونلاقي ال Left وال Right Nodes بتوعها بـ null ساعتها بقوله رجعلي 0 عشان ال null مبيتعدش إنه Level أكيد.

height(t(L, Root, R), H):-

اديتلاها tree ليها Right و Left Nodes ومستنية H اللي هيرجع فيه ال Height النهائي لل tree دي.

إزاي هحسب ال Height لل tree بقى وأنا عندي 2 Paths.. Left Tree كاملة و Right Tree، وأنا مش عارفة انهي واحدة أطول؟!! فالحل إني أجيب ال Height بتاع دي وبتاع دي واقارنهم ببعض وأخد ال maximum فيهم يبقى هوا ده Height ال tree كلها.

عشان كده عرفت Predicate اسمها max هيا اللي هتقارن بين ال 2 Heights وترجعلي ال maximum فيهم.

height(L, HL),

بنده على نفسي بال Left Node وهيا هتشتغل مع نفسها وتجبلي ال Height بتاع ال Tree بتاعتها.

height(R, HR),

نفس الكلام بس علي ال Right Node.

max(HR, HL, M),

max اخدت ال Height لل Right Node (HR) وال Height لل Left Node (HL)، وهترجعلي M فيه ال maximum بين الإتنين.

H = M +1.

بعدين باجمع على ال M واحد عشان خاطر ال Root بتاعي، وتبقى هيا دي ال H بتاعت ال tree كلها.

Root هنا هتدي Warning إنها Unused برضو فابقوا شيلوها أنا سايباها عشان أعرف أشرح بس.

Maximum Height:

max(HR, HL, HR):-

HR > HL, !.

max بأديها ال HR وال HL وهترجعلي ال HR لو هوا أكبر.

max(_, HL, HL).

ولو HL أكبر هترجعهولي في ال Case التانية.

Goal

height(t(t(t(null, 1, null), 3, t(null, 4, null)), 5, t(t(null, 7, null), 8, t(t(null, 9, null), 11, null))), X).

Output:

X=4

1 Solution

تمام هما فعلا 4 Levels.. ولو جربت تشيل Nodes وت Check بتشتغل صح برضو.

Assignment 2:

Make a predicate "linearise (T, L)." which takes a tree and returns a list contains nodes(values) of that tree.

Deletion from Trees:

1- Delete Leaf:

هنعمل Predicate اسمها delete_leaf بأديها tree و ال integer اللي عاوزة اعمله Delete من ال tree دي وبترجعلي new tree مفهاش ال element ده.

Predicates

nondeterm delete_leaf(tree, integer, tree).

عرفت delete_leaf بتاخد tree وال leaf اللي عاوزة اشيله وبترجع نفس ال tree من غير ال leaf.

وكاتبة هنا nondeterm عشان فيه اتنين delete_leaf في ال Clauses ليهم نفس ال Arguments عشان يدخل في الإتنين يعني.

Clauses

delete_leaf(t(null, X, null), X, null).

Base Step لما X ت Match مع leaf من Leaves ال tree فتشيله وترجع null بداله.

delete_leaf(t(L, Root, R), X, t(NL, Root, R)):-

X < Root, !,

delete_leaf(L, X, NL).

أول Recursive Step لو X أقل من ال Root معني كده إنه leaf في ال Left Tree، فبنده على نفسي بال L.

delete_leaf(t(L, Root, R), X, t(L, Root, NR)):-

delete_leaf(R, X, NR).

تاني Recursive Step لو X أكبر من ال Root هينده على نفسه بال Right Tree بس عشان أكيد هيا جواها.

- اللي بالاحمر ده، أنا ليه بارجع tree فيها كل ده مش ال NL أو ال NR بس؟!
عشان NR أو NL (على حسب دخل في انهي Case منهم) دي Branch واحد من ال tree مش ال tree كلها، وأنا عاوزاه يرجعلي ال New Tree كلها بعد مايتشال منها ال Leaf ده.. فعشان كده بخُط NL أو NR كجزء من ال tree مع ال Root وال R أو ال L القدام (عشان هما نفسهم ماحصلهمش تغيير) كده ال New Tree كملت بالقديم والجديد.

Goal

delete_leaf(t(t(t(null, 1, null), 3, t(null, 4, null)), 5, t(t(null, 7, null), 8, t(t(null, 9, null), 11, null))), 7, NT).

Output:

NT=t(t(t(null,1,null),3,t(null,4,null)),5,t(null,8,t(t(null,9,null),11,null)))

1 Solution

فعلا عمل Delete لل Leaf Node اللي هيا 7 وساب الباقي..

لو قلتله Delete ل Node مش Leaf هي Fail ويرجعلك No Solution، عشان هوا ال Base Step بتاعته إن ال Left Node وال Right Node يبقوا بـ null (شرط ال Leaf) فأي Node عادية مش Leaf هيبقى فيه Matching مع ال Case دي فعشان كده بيخرج ويديك No Solution.. حتى جربوها.

2- Delete Minimum:

هنعمل Predicate اسمها delmin بتاخد tree وترجع integer (minimum value found)
وترجع ال tree بعد ما إتشالت منها ال Value دي.
مادام أنا عاوزة ال Minimum Value يبقى هروح على ال Left Tree لل Leaf بتاعها هتبقى هيا دي ال minimum.. بلاعملها Delete وأرجع ال Tree من غيرها.

Predicates

nondeterm delmin(tree, integer, tree).

Clauses

delmin(t(null, X, R), X, R):- !.

Base Step أما X ت Match مع ال Root ل Tree ال Left Node بتاعتها بـ null معني كده إني وصلت لآخر Node في ال Tree دي فتبقى هيا أقل Value اللي أنا عاوزها.. فبعملها Delete وأرجع Tree فيها ال Right Nodes بس.

delmin(t(L, Root, R), X, t(NL, Root, R)):-
delmin(L, X, NL).

Recursive Step هتفضل تدخل في ال Left Tree لحد ماتوصل لل Base Step وتعملها Delete وبعدين ترجع ال NL مع ال Root وال Right Node زي مهما.

Goal

delmin(t(t(t(null, 1, null), 3, t(null, 4, null)), 5, t(t(null, 7, null), 8, t(t(null, 9, null), 11, null))), X, NT).

Output:

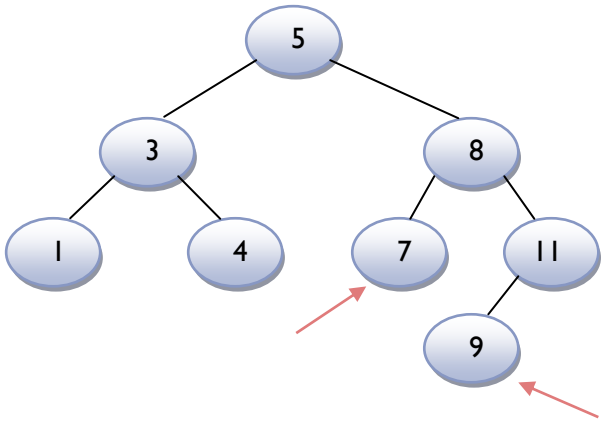
X=1,
NT=t(t(null,3,t(null,4,null)),5,t(t(null,7,null),8,t(t(null,9,null),11,null)))
1 Solution

عمل Delete لل 1 عشان هوا ال Minimum في ال Tree دي ورجعلي قيمة ال 1 مع ال New Tree من غير ال 1.

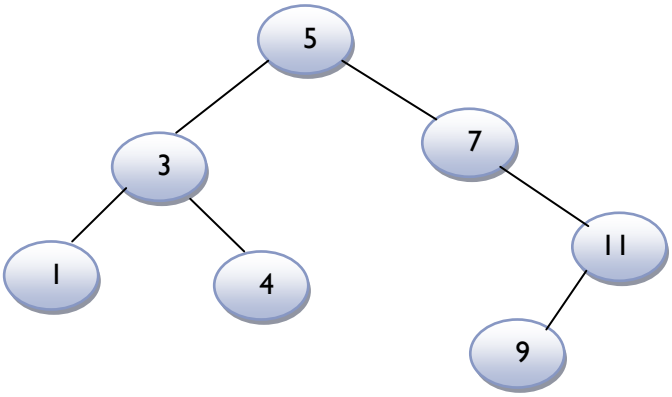
3- Delete Any Node:

احنا معملناش إزاي ن Delete أي Node في ال Tree بس اخدنا الفكرة زي بتاعت المحاضرة، هاكتب الفكرة وممكن انتوا تعملوها مع نفسكوا..

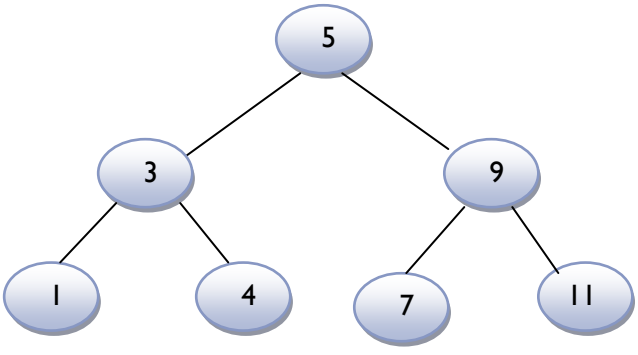
لو عندي Tree زي دي وعايزة أعمل Delete لل Node اللي فيها 8، طيب وال Nodes اللي تحتها هتروح فين.. لازم اختار Node منهم أحطها بدل ال 8 عشان بقيت ال Nodes متضيعش وال Node لازم تحقق شروط ال Binary Dictionary إن ال Left Node تبقى أقل من ال Root وال Right أكبر من ال Root. قدامي حلين لإما أدخل في ال Left Tree لل 8 وأخد أكبر Node وأحطها مكان ال 8، لإما أدخل في ال Right Tree لل 8 وأخد أصغر Node فيها.



1. ال Tree بعد متشالت منها ال 8 وحطيت بدالها 7:



2. هنا شيلت منها ال 8 وحطيت بدالها 9:



حاولوا تعملوا delete مع نفسكوا بقى، بتاخد Tree وinteger (ال Node اللي عايز تعملها Delete) وبترجع ال New Tree من غير ال Node دي.

Assignments 1 & 2:

1- Use merge and split predicates to make "merge sort" predicate.

2- Make a predicate "linearise (T, L)." which takes a tree and returns a list contains nodes(values) of that tree.

Good Luck