Dayou Du
N13825075
dd2645@nyu.edu

Homework 10
Fundamental Algorithms

2017-12-05

Ex 11.anotherDP

Let $Sum[i]$ be the sum of the first elements i

$$Sum[i] = \begin{cases} 0 & ,i = 0 \\ Sum[i-1] + S[i] & ,i > 0 \end{cases} \tag{1}$$

Let $MinSquareSum[i]$ be the minimum sum of the square of the sum of the numbers in each subsequence in the first i numbers.

$$MinSquareSum[i] = \begin{cases} 0 & ,i = 0 \\ min_{0 \le j < i}\{MinSquareSum[j] + (Sum[i] - Sum[j])^2\} & ,0 < i \le n \end{cases} \tag{2}$$

The answer is $MinSquareSum[n]$

Ex 11.50

a)

$$Ubest(w) = w.dat + \begin{cases} 0 & ,w\ is\ a\ leaf \\ \sum_{v\ in\ Adj[w]} Kbest(v) & ,w\ is\ not\ a\ leaf \end{cases} \tag{3}$$

$$Kbest(w) = \begin{cases} 0 & ,w\ is\ a\ leaf \\ \sum_{v\ in\ Adj[w]} max\{Ubest(v), Kbest(v)\} & ,w\ is\ not\ a\ leaf \end{cases} \tag{4}$$

The answer is $max\{Ubest(root), Kbest(root)\}$

b)

Use additional two fields for each vertex to store the result of $Ubest$ and $Kbest$. Don't calculate again for the subproblems if these fields are calculated.

c)

Algorithm 1 Ex 11.50c

```
Global $children[1..n], Data[1..n]$
function $Ubest(i)$
    if $Data[i].Bbest \neq Nil$ then
        return $Data[i].Bbest$
    $Data[i].Bbest \leftarrow Data[i].dat$
    for each $j \in children[i]$ do
        $Data[i].Bbest \leftarrow Data[i].Bbest + Kbest(j)$
    return $Data[i].Bbest$
function $Kbest(i)$
    if $Data[i].Gbest \neq Nil$ then
        return $Data[i].Gbest$
    $Data[i].Gbest \leftarrow 0$
    for each $j \in children[i]$ do
        $Data[i].Gbest \leftarrow Data[i].Gbest + max\{Kbest(j), Ubest(j)\}$
    return $Data[i].Gbest$
```

Ex 5.Graph

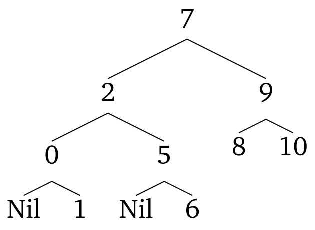
Algorithm 2 Ex 5.Graph

```
procedure $BuildAdjList(n, Edges[1..|E|])$
    Crate $Temp[1..n]$ where each $Temp[i]$ is an empty linked list
    for each $i \in [1, |E|]$ do
        $(u, v) \leftarrow Edges[i]$
        put $u$ at the end of $Temp[v]$
    Crate $Adj[1..n]$ where each $Adj[i]$ is an empty linked list
    for each $i \in [1, n]$ do
        for each $u \in Temp[i]$ do
            put $i$ at the end of $Adj[u]$
```

Ex 5.40a

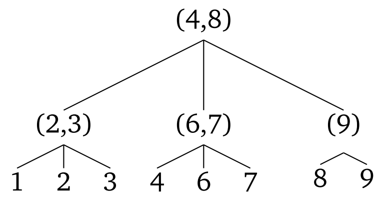
$$T(n = 2^l + r) = (l + 1)n + 2r \tag{5}$$

Ex 6.5

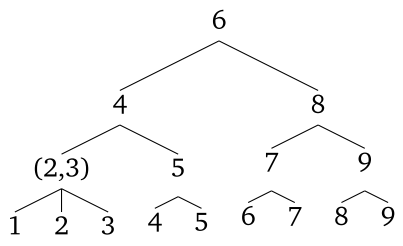


Ex 6.insertdelete

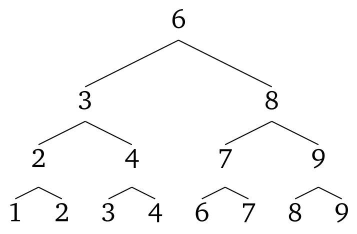
a



b



c



Ex 6.10a

On the path from the root of the 23tree down to the vertex that would be come the parent of the leaf-records x, each of the vertex on the path have 3 children

Ex 6.range

Use an additional $.count$ in each internal vertex v to store the number of leaves in the subtree rooted at v
Insert: Insert the node as the same insertion process in standard 23tree, and also adjust the $.count$ fields as needed. (If there are structure changes, count the $.count$ field for the changed nodes by accessing the $.count$ field of its' children.)
Locate: Search the node as the same search process in standard 23tree.
Delete: Delete the node as the same deletion process in standard 23tree, and also adjust the $.count$ fields as needed. (If there are structure changes, count the $.count$ field for the changed nodes by accessing the $.count$ field of its' children.)
CountLessThan: Search the record with key x. On the path from the root of the 23tree down to the vertex of key x(or the vertex have the largest value under x), calculate the sum of the $.count$ fields of the left siblings for all nodes on the path.

Ex 6.intervals

Store a pair(interval) as the element in 23tree, use the lower bound as the primary key and use the upper bound as the secondary key: for pair (a, b) and (c, d), $(a, b) < (c, d)$ if and only if $a < c$ or ($a == c$ and $b < d$).
The **Insert,Locate and Delete** operations are just the same as the corresponding process in the standard 23tree, just use the interval compare function using a primary key and a secondary key.
The interval compare function is still $\Theta(1)$, thus these operations are still $O(logn)$, which is the same as the original 23tree.

Ex 6.intervalstabs

Use two 23tree as implemented in Ex 6.intervals: One use the lower bound as the primary key(name it $T1$), the other use the upper bound as the primary key(name it $T2$). $T1$ and $T2$ also maintains the $.count$ fields as we implemented in Ex 6.range
Then $Stab(x) = T1.CountLessThan(x, +\infty) - T2.CountLessThan(x, x)$

Ex 8.qqq

a)

$$Sarray = \begin{bmatrix} R & R & +\infty & +\infty \\ +\infty & W & W & +\infty \\ +\infty & +\infty & B & B \\ +\infty & +\infty & +\infty & +\infty \end{bmatrix}$$

$Pcost[i][j] = Scost[i][j+3n], i, j \leq n$

b)

$64cn^3 + O(n^2)$

c)

Algorithm 3 Ex 8.qqqc

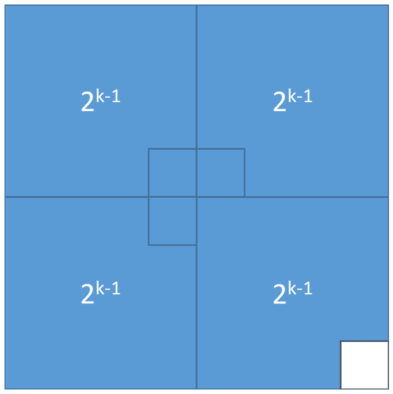
```
Input: $Ecost$, the original cost array, and colors of the edges
Output: $Pcost$, the cost array that satisfied $R^+W^+B^+$
1: procedure $8.qqqc(Ecost, R, W, B; ; Pcost)$
2:     build 3 $n \times n$ arrays from $Ecost$ with same color of edges: $Rsingle$, $Wsingle$ and $Bsingle$
3:     set all $[i, i]$ entries in $Rsingle$, $Wsingle$ and $Bsingle$ to $\infty$
4:     Run $FW$ algorithm independently on $Rsingle$, $Wsingle$ and $Bsingle$, get 3 $n \times n$ arrays $Rmulti$,$Wmulti$ and $Bmulti$
5:     set all $[i, i]$ entries in $Rmulti$, $Wmulti$,$Bmulti$ to 0
6:     create empty temporal $n \times n$ cost array $Temp$
7:     $Best2Step(n, Rmulti, Rsingle, Pcost)$
8:     $Best2Step(n, Pcost, Wmulti, Temp)$
9:     $Best2Step(n, Temp, Wsingle, Pcost)$
10:    $Best2Step(n, Pcost, Bmulti, Temp)$
11:    $Best2Step(n, Temp, Bsingle, Pcost)$
```

d)

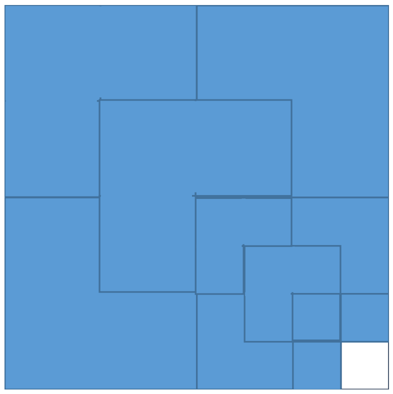
$(3 + 5) * (cn^3 + O(n^2)) = 8cn^3 + O(n^2)$

Ex 11.14

a)



b)



c)

For the elegant solution: The basic idea of elegant solution is use the smaller situations : $2^i \times 2^i$ block with a corner missed, to construct the larger one.
For the 2^k board, we can still split the board into 4 half-sized boards, and rotate the board to let the missed block in to the bottom right half-sized board. We can do the same thing as in question(a) to the other three boards.
Then the problem becomes a smaller problem that a random block was missed in a 2^{k-1} board, do the same split and rotate thing. The work will be done.

For the clunky solution: The basic idea is form a series of larger 'L' blocks(i.e. the same shape as the basic Tri-ominoes, but the linear size is 2^i larger) using smaller 'L' blocks.
For the 2^k board, we can still split the board into a big 'L' shape and a 2^{k-1} board with a random block is missed. Then we do recursion on the 2^{k-1} board.

Ex 8.onemore

a)

$r^*w^*b^+$

b)

r^+

c)

r^*w^+

Ex 8.last

a)

rwb^*

b)

$r^+w^*b^+|r^+b^*$

c)

$(r^*w^*b^+)^+$