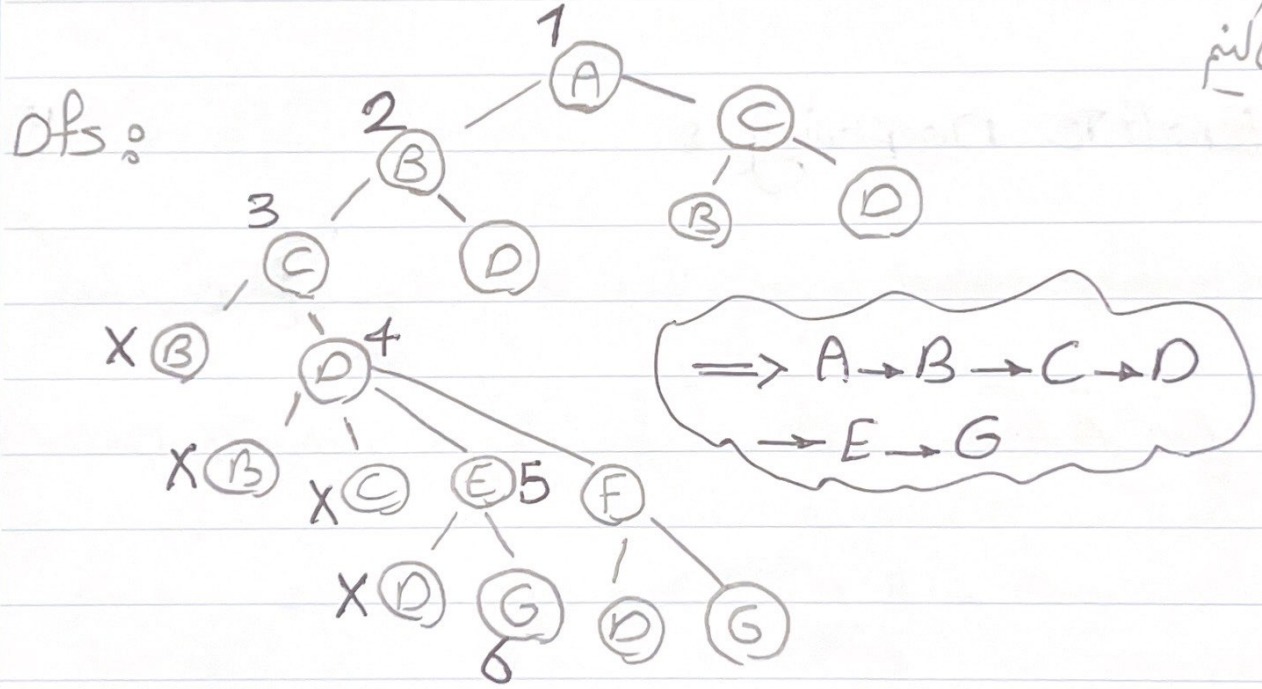
۱-

بخش تئوری تمرین سری اول ؛ ۹۹۵۲۱۰۸۲ - بابک اشعاری

۱- ابتدا برای هر روش درخت رسم می‌کنیم و ترتیب نودها را مشخص می‌کنیم

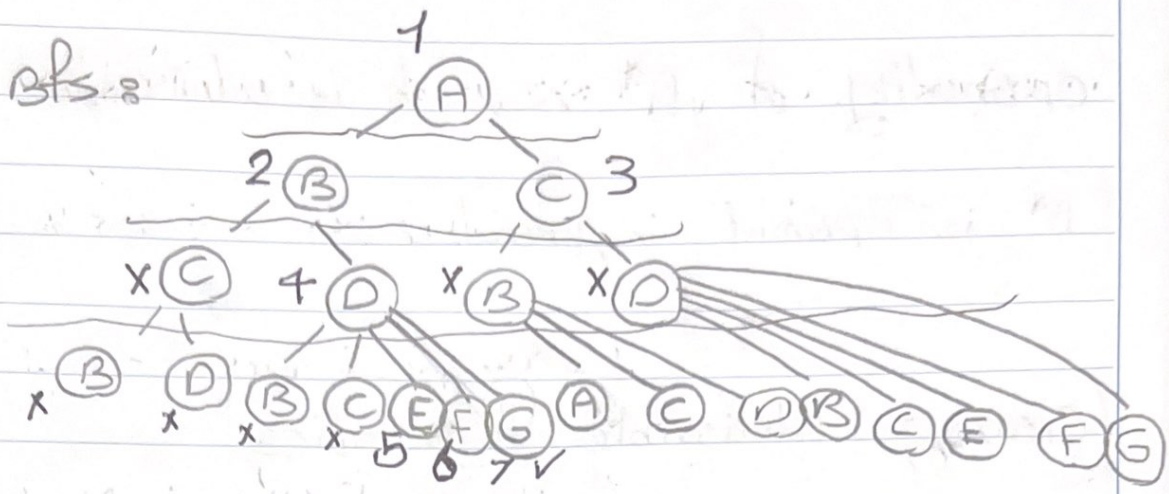


PAPA

Date : / /

Sabject :

Bfs :



Expand : $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$

Path : $P.G = D \rightarrow P.D = B \rightarrow P.B = A$

$\Rightarrow$ Path : $A \rightarrow B \rightarrow D \rightarrow G$

۲- Iterative Deepening :

یکی از روش های ناآگاهانه جست وجو، (Uniform search) می باشد به طوری که با ترکیب دو الگوریتم سریع Iterative Deepening dfs و bfs از مزایای هر دو استفاده می کند. الگوریتم آن هم

Date : / /
Sabject :

به این شکل است که در هر مرحله الگوریتم Dfs را با عمق های محدود اجرا می‌کند تا به goal برسد.

۱- اجرای Dfs تا عمق محدود ۱

۲- اجرای Dfs تا عمق محدود ۲

۳- اجرای Dfs تا عمق محدود ۳ (الی آخر)

اگر تعداد نودهای هر branch را b در نظر بگیریم و عمق درخت را d باشد در عمق اول با b^0 تا نود داریم، در عمق بعدی b^0+b^1 نود داریم (در هر بار اجرای نودهای تکراری دفعات قبل نیز دوباره محاسبه می‌شوند که این یکی از معایب این روش است)

و در آخر:

$1+b+b^2+b^3+\dots b^d \Rightarrow$ Time complexity : $O(b^d)$

در نتیجه از نظر زمانی این روش تقریباً با Bfs برابری می‌کند

PAPA

Date : / /
Sabject :

(ما الی ۲۵ درصد بنا‌ظر order های پایین بیشتر است) واز آن جایی که هر بار مقادیر دفعات قبلی پاک می‌شوند از نظر حافظه عملکرد مناسب dfs دارد.

کاربرد این روش در مواقعی است که ما می دانیم جواب ما در لول های بالایی درخت وجود دارد و با محدودیت حافظه داریم و به همین دلیل نمی‌توانیم از Bfs استفاده کنیم، درنتیجه از این روش استفاده می‌کنیم.

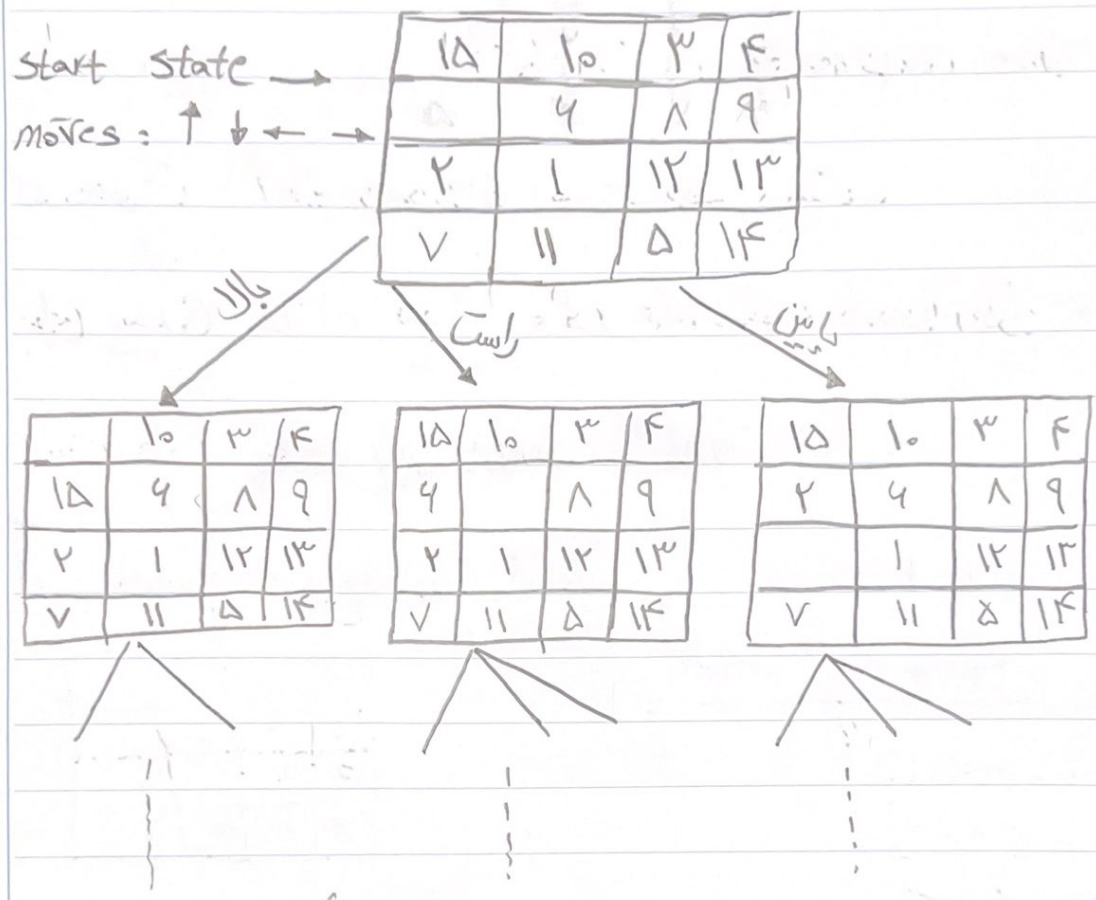
۳- اگر بتوانیم تمامی حالات ممکن برای پازل را به صورت درخت در بیاوریم (به طوری که هر بچه از مادر خود تشکیل می‌شود) آنگاه می‌توانیم پیمایش روی درخت را توسط الگوریتم های Bfs یا dfs انجام دهیم.

PAPA

Date : / /

Sabject :

حال برای ایجاد درخت حالات از خانه‌ی خالی داخل بازل استفاده می‌کنیم. و هر بچه‌ی حالت شروع، حرکت خانه‌ی خالی به‌سمت بالا، پایین، چپ، راست است. برای مثال اگر حالت شروع بازل زیر باشد:



برای هر حالت (node) همسایه‌ها را با حرکت بخش خالی می‌سازیم و الگوریتم‌های bfs و dfs را تا زمانی که به حالت

Date : / /

Sabject :

پایانی برسم، ادامه می‌دهم.

پیچیدگی زمانی: با توجه به تعداد خودها در هر مرحله و عمق درخت اگر فرض کنیم branch factor = b و depth = d باشد داریم:

$$\text{Time complexity} = O(b^d)$$

که بصورت Exponential است و بهینه نیست.

برای پیچیدگی حافظه نیز dfs عملکرد بهتری دارد و برابر طول درخت است ولی bfs برابر بیشترین عرض درخت است.

برای محاسبه branching factor نیز داریم:

$$b = \frac{4\times 2 + 8\times 3 + 4\times 4}{14} = 3$$

$$T = O(3^d)$$

X	○	○	X
○	△	△	○
○	△	△	○
X	○	○	X

PAPA