General Algorithm for graph search

Let fringe be a list containing the initial state

Let closed be initially empty

Loop

    If fringe is empty return failure

    Node<- remove_first(fringe)

    If **Node** is goal

        Then return the path from initial state to Node S

    Else put **Node** in closed

        Generate all successors of **Node** S

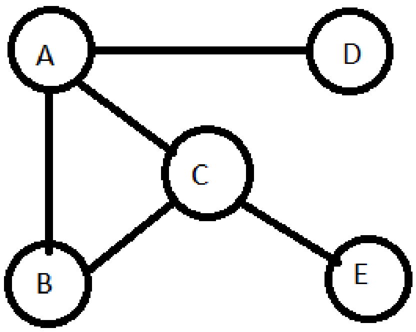
        For all nodes m in S

            If m is not in closed

            Merge m into fringe

End Loop

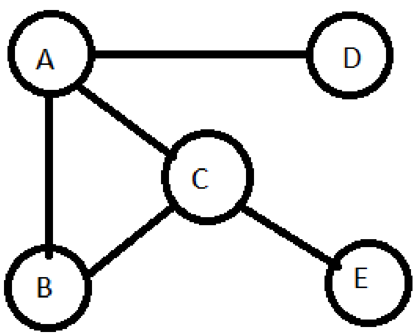
BFS Graph traversal



A The start node is in Open

Open	Closed	Next move
A	-------	A removed from open, add A in closed, add Successors of A in back of the Open (BCD)
BCD	A	Remove B from front of open, add it in closed, add successors of B at the back of the Open (A is in visited so omit, add C in the start of the queue)
CDC	AB	Remove C from front of open, add it in closed, add successors of C (A is visited so omit, B Visited so omit and add E in the end of the queue) at the back of the Open
DCE	ABC	Remove D from front of open, add it in closed, add successors of D at the back of the Open (no successors found)
CE	ABCD	Remove C from the front of open, it exists in closed, no action
E	ABCD	Remove E from the front of open, C is successor but is present in closed, no successors to be added in open, put E in closed
-----	**ABCDE**	--------

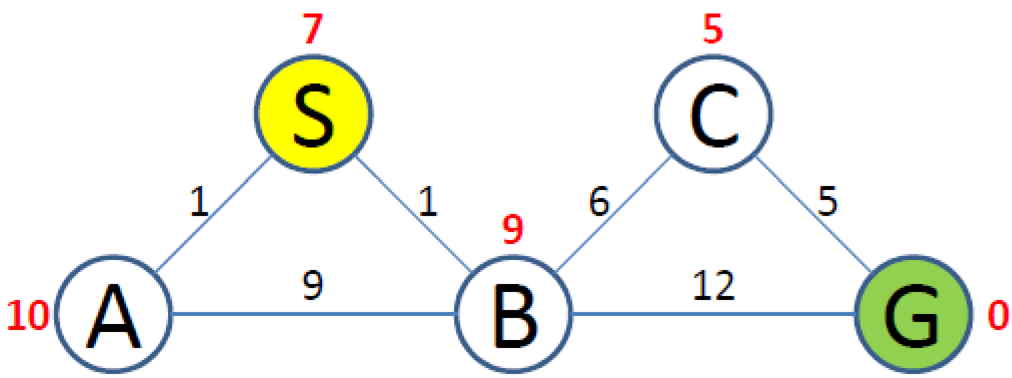
DFS Graph traversal

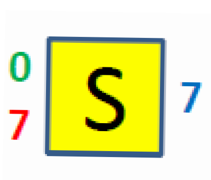


A The start node is in Open

Open	Closed	Next move
A	-------	A removed from open, add A in closed add Successors of A in the start of Open,
BCD	A	Remove B from front of open, add it in closed, add successors of B at the start of the Open (A is in visited so omit, add C in the start of the queue)
CCD	AB	Remove C from front of open, add it in closed, add successors of C (A is visited so omit, B Visited so omit and add E in the start of the queue)
ECD	ABC	Remove E from front of open, add it in closed, add successors of E at the start of the Open (C is already visited so omit)
CD	ABCE	Remove C from the front of open, it exists in closed, no action
D	ABCE	Remove D from the front of open, no successors to be added in open(as A in closed) , put D in closed
-----	**ABCED**	Open Empty

A* Examples



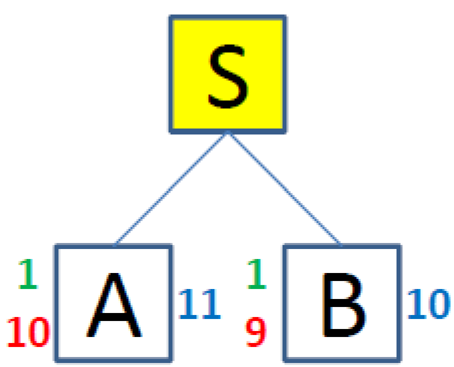


Expanded Node	Open
--	S7

Step 1: Start node S, Successors A & B

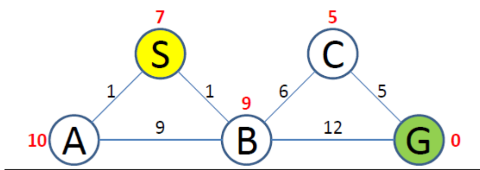
S-A => f(A) = 1+10=11

S-B => f(B)= 1+9=10



Expanded Node	Open

--	S7
S7	B10 A11



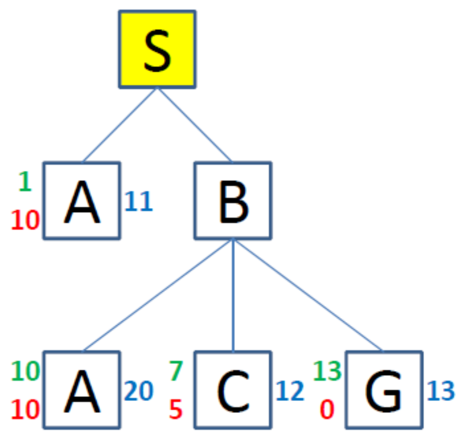
Step 2: S-B , Successors of B are A, C, G

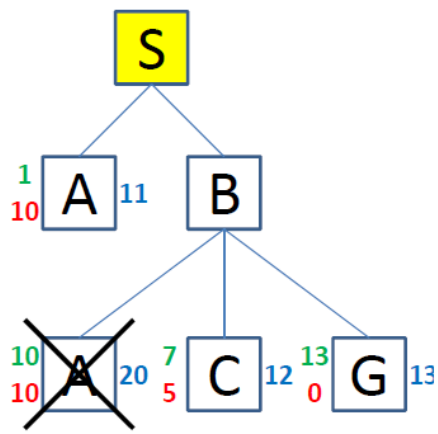
S-B-A => f(A) = (1+9)+10=20....Discard

S-B-C => f(C)= (6+1)+5=12

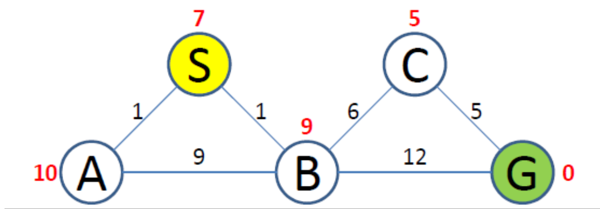
S-B-G => f(G)= (1+12)+0=13

S-A path from step 1 chosen as min f(n)





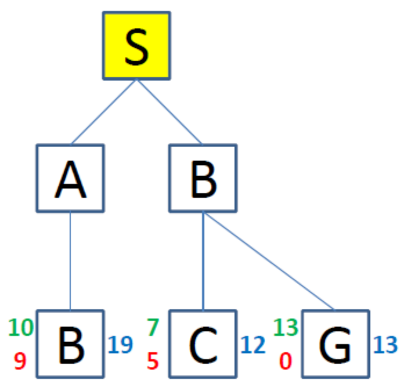
Expanded Node	Open
--	S7
S7	B10 A11
B10	A11 C12 G13

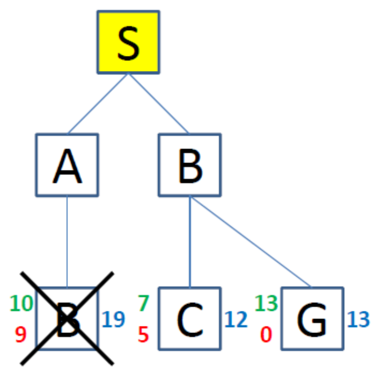


Step 3: S-A , Successors is B

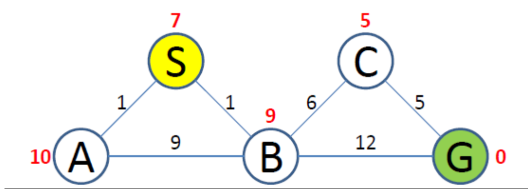
S-A-B => f(B) = (1+9)+9= 19 ....Discard

S-B-C chosen as min f(n)





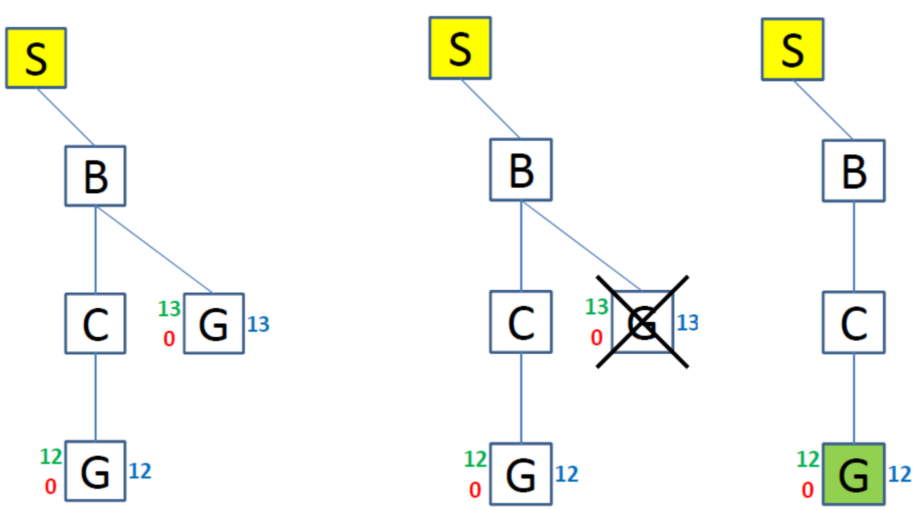
Expanded Node	Open
--	S7
S7	B10 A11
B10	A11 C12 G13
A11	C12 G13



Step 4: S-B-C , Successors is G

S-B-C-G => f(G) = (1+6+5)+0= 12 ......Discard G13(SBG)

SBCG optimal path from S to G



Expanded Node	Open
--	S7
S7	B10 A11
B10	A11 C12 G13
A11	C12 G13
C12	G12

G12	EMPTY

Perform the A* Algorithm on the following figure. Explicitly write down the queue at each step.