Advanced Algorithms Analysis and Design

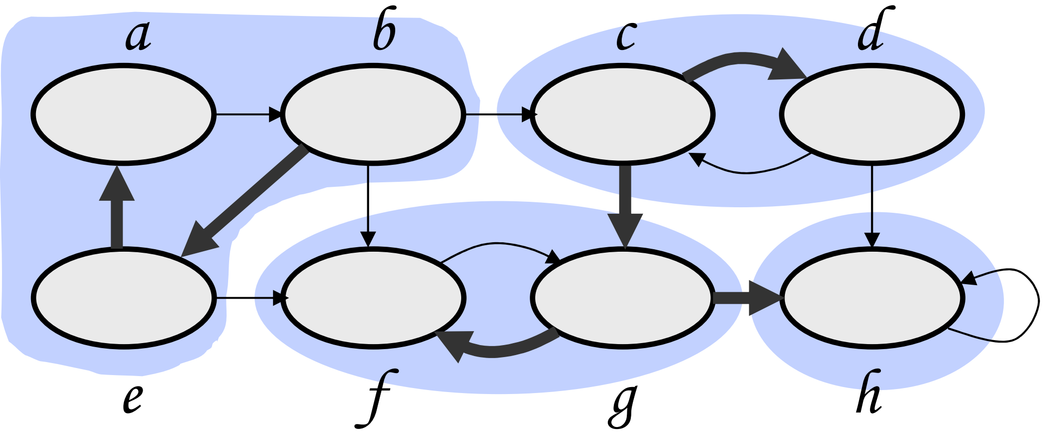
By

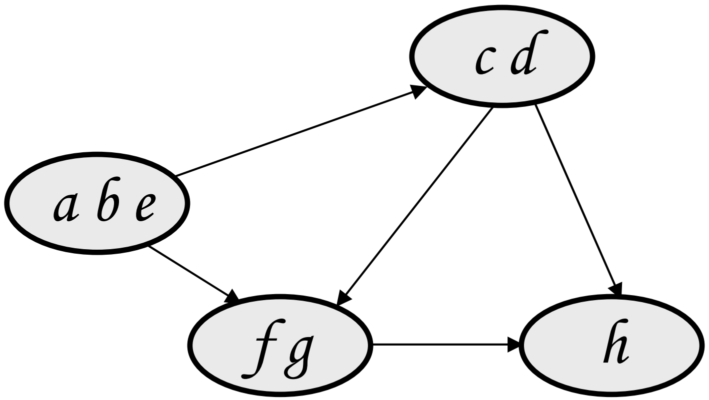
Nazir Ahmad Zafar

Lecture No 31

Backtacking and Branch & Bound Algorithms

Component Graph





- The **component graph** $G^{SCC} = (V^{SCC}, E^{SCC})$
  - $V^{SCC} = \{v_1, v_2, \ldots, v_k\}$, where v_i corresponds to each strongly connected component C_i
  - There is an edge $(v_i, v_j) \in E^{SCC}$ if G contains a directed edge (x, y) for some $x \in C_i$ and $y \in C_j$
- The component graph is a DAG Lemma

Lemma 1

Let C and C' be distinct SCC's in G

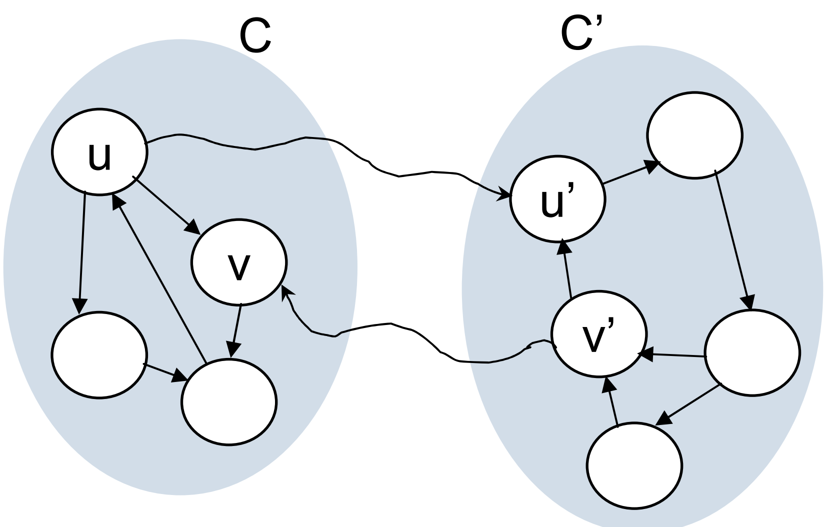
Let u, v $\in$ C, and u', v' $\in$ C'

Suppose there is a path u $\leadsto$ u' in G

Then there cannot also be a path v' $\leadsto$ v in G.

Proof

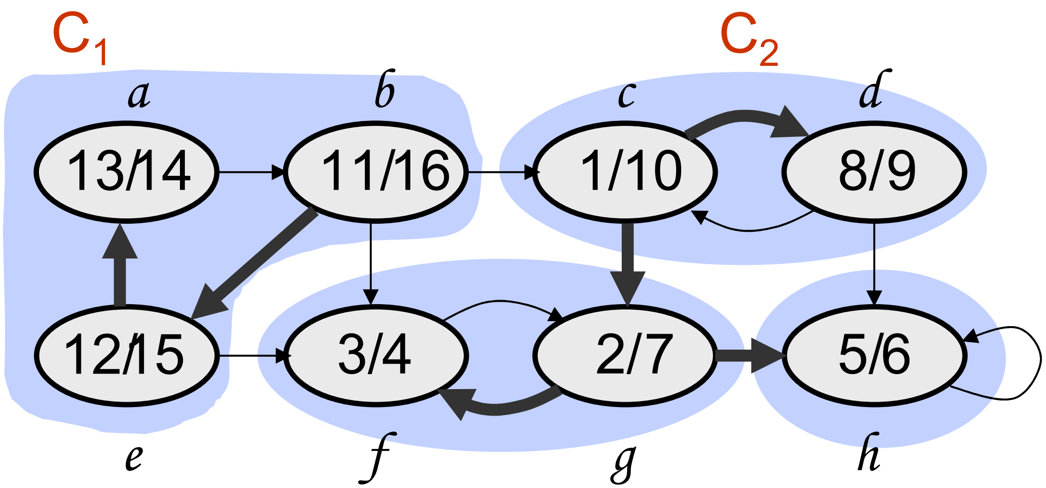
- Suppose there is path v' $\leadsto$ v
- There exists u $\leadsto$ u' $\leadsto$ v'
- There exists v' $\leadsto$ v $\leadsto$ u
- u and v' are reachable from each other, so they are not in separate SCC's: contradiction!



Notations

Notations: Vertices to SCC

- d and f times of vertices of SCC
- Let $U \subseteq V$, a SCC
  - $d(U) = \min_{u \in U} \{ d[u] \}$ (earliest discovery time)
  - $f(U) = \max_{u \in U} \{ f[u] \}$ (latest finishing time)



$d(C_1)=11$
$f(C_1)=16$

$d(C_2)=1$
$f(C_2)=10$

C_3
$d(C_3)=2$
$f(C_3)=7$

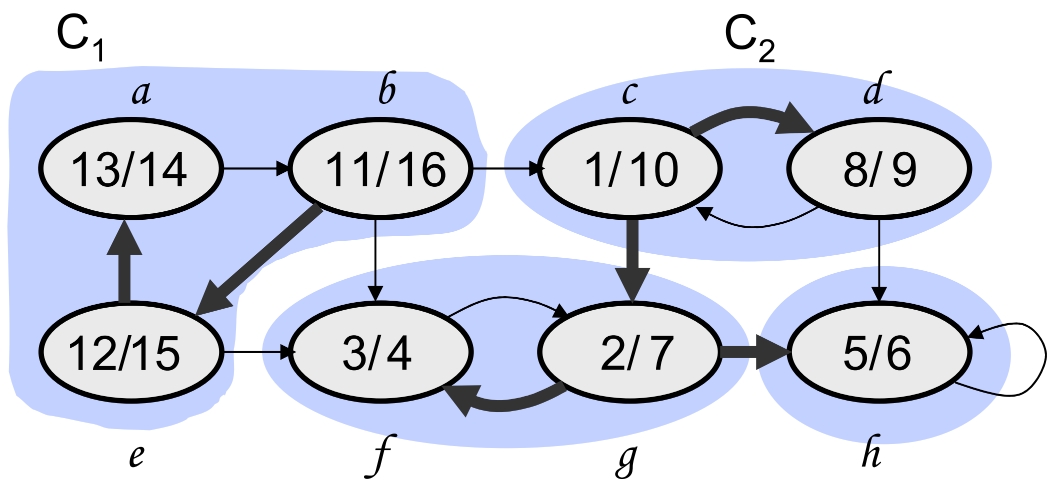
C_4
$d(C_4)=5$
$f(C_4)=6$

Lemma

Lemma 2

- Let C and C' be distinct SCCs in a directed graph G = (V, E). If there is an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$ then $f(C) > f(C')$.

$d(C_1) = 11$
$f(C_1) = 16$



$d(C_2) = 1$
$f(C_2) = 10$

C_3
$d(C_3) = 2$
$f(C_3) = 7$

C_4
$d(C_4) = 5$
$f(C_4) = 6$

Lemma 2

Proof

- Consider C1 and C2, connected by edge (u, v)
- There are two cases, depending on which strongly connected component, C or C', had the first discovered vertex during the depth-first search

Case 1

- If $d(C) < d(C')$, let x be the first vertex discovered in C. At time $d[x]$, all vertices in C and C' are white.
- There is a path in G from x to each vertex in C consisting only of white vertices.
- Because $(u, v) \in E$, for any vertex $w \in C'$, there is also a path at time $d[x]$ from x to w in G consisting only of white vertices: $x \leadsto u \longrightarrow v \leadsto w$.

Lemma 2 (Cont..)

- By the white-path theorem, all vertices in C and C' become descendants of x in the depth-first tree. By Corollary, $f[x] = f(C) > f(C')$.

Case 2

- $d(C) > d(C')$ (supposition)
- Now (u, v) $\in$ E, where u $\in$ C and v $\in$ C' (given)
- Let y be the first vertex discovered in C'.
- At time $d[y]$, all vertices in C' are white and there is a path in G from y to each vertex in C' consisting only of white vertices.

Lemma (Cont..)

- By the white-path theorem, all vertices in C' become descendants of y in the depth-first tree, and by Corollary, $f[y] = f(C')$.
- At time $d[y]$, all vertices in C are white. Since there is an edge (u, v) from C to C', Lemma implies that there cannot be a path from C' to C.
- Hence, no vertex in C is reachable from y.
- At time $f[y]$, therefore, all vertices in C are still white.
- Thus, for any vertex $w \in C$, we have $f[w] > f[y]$, which implies that $f(C) > f(C')$.

Corollary

Corollary

Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$. Then $f(C) < f(C')$

Proof

- Since $(u, v) \in E^{\mathrm{T}}$, we have $(v, u) \in E$.
- Since strongly connected components of G and G^{T} are same, Lemma implies that $f(C) < f(C')$.

Correctness Theorem

Theorem: Correctness of SCC Algorithm

STRONGLY-CONNECTED-COMPONENTS (*G*) correctly computes SCCs of a directed graph *G*.

Proof

- We argue by induction on number of DF trees of G^T that "vertices of each tree form a SCC".
- The basis for induction, when $k = 0$, is trivial.
- Inductive hypothesis is that, first k trees produced by DFS of G^T are strongly connected components.
- Now we prove for $(k+1)^{st}$ tree produced from G^T, i.e. vertices of this tree form a SCC.
- Let root of this tree be u, which is in SCC *C*.
- Now, $f[u] = f(C) > f(C')$, $\forall$ C' yet to be visited and $\neq C$

Theorem: Correctness of SCC Algorithm

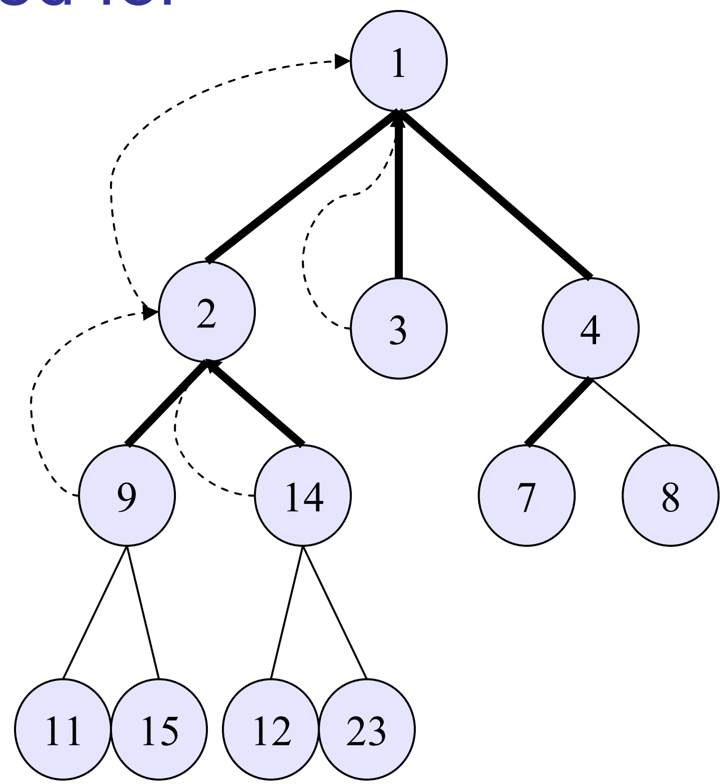
- By inductive hypothesis, at the time search visits u, all other vertices of C are white.
- By white-path theorem, all other vertices of C are descendants of u in its DF tree.
- Moreover, by inductive hypothesis and by Corollary above, any edges in G^T, that leave *C* must be, to SCCs that have already been visited.
- Thus, no vertex in any SCC other than *C* will be a descendant of *u* during the DFS of G^T.
- Thus, vertices of DF tree in G^T rooted at *u* form exactly one SCC.

Today Covered

- Why backtracking?
- What is backtracking?
- Backtracking
  - Solution Spaces
  - Knapsack Problem
  - The Queens Problem
- Branch and bound technique
  - Assigning Task to Agents

Why BackTracking?

- When the graph is too large
  - Depth and breadth-first techniques are infeasible
- In this approach if node searched for
  - is found out that cannot exist in the branch then
  - return back to previous step and continue the search to find the required node



- What is backtracking?

What is BackTracking

- Backtracking is refinement of Brute Force approach
- It is a technique of constraint satisfaction problems
- Constraint satisfaction problems are with complete solution, where elements order does not matter.
- In backtracking, multiple solutions can be eliminated without examining, by using specific properties
- Backtracking closely related to combinatorial search
- There must be the proper hierarchy in produces
- When a node is rejected, whole sub-tree rejected, and we backtrack to the ancestor of node.
- Method is not very popular, in the worst case, it takes an exponential amount of time to complete. (S. Space)

Solution Spaces

- Solutions are represented by vectors ($v_1, \ldots, v_m$) of values. If S_i is the **domain** of v_i, then $S_1 \times \ldots \times S_m$ is the **solution space** of the problem.
- **Approach**
  - It starts with an empty vector.
  - At each stage it extends a partial vector with a new value
  - Upon reaching a partial vector ($v_1, \ldots, v_i, v$) which can't represent a partial solution, the algorithm backtracks by removing the trailing value from the vector, and then proceeds by trying to extend the vector with alternative values. (Algorithm)

General Algorithm: Solution Spaces

ALGORITHM $try(v_1,...,v_i)$

IF $(v_1,...,v_i)$ is a solution

THEN RETURN $(v_1,...,v_i)$

FOR each v DO

IF $(v_1,...,v_i,v)$ is acceptable vector

THEN

sol = $try(v_1,...,v_i,v)$

THEN RETURN sol

Knapsack

Knapsack: Feasible Solutions

- Partial solution is one in which only first *k* items have been considered.
  - Solution has form $S_k = \{x_1, x_2, \ldots, x_k\}$, $1 \leq k < n$.
  - The partial solution S_k is feasible if and only if

$$\sum_{i=1}^{k} w_i x_i \leq C$$

  - If S_k is infeasible, then every possible complete solution containing S_k is also infeasible.

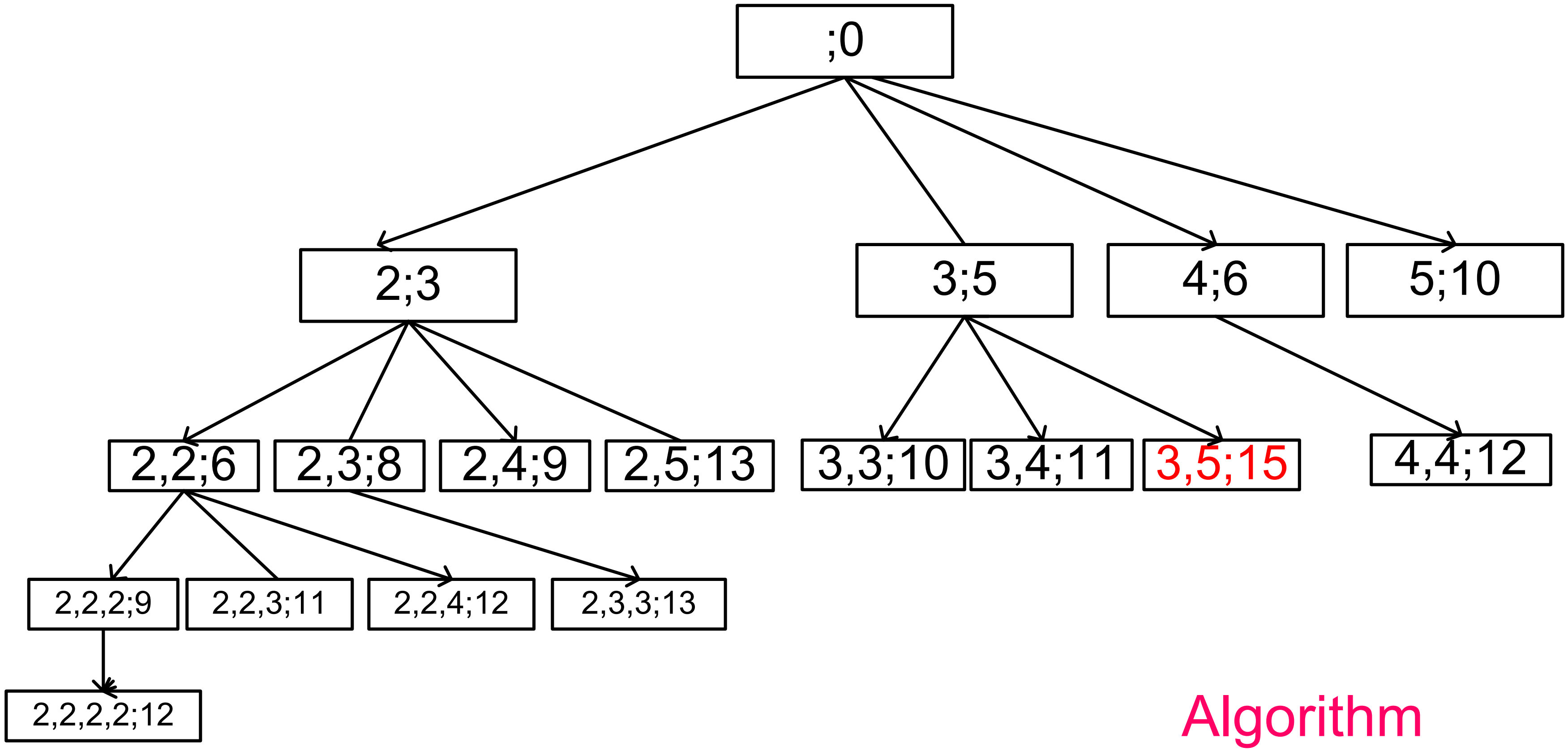
Knapsack Example: Backtracking

Maximum Capacity = 8

i	1	2	3	4
v_i	3	5	6	10
w_i	2	3	4	5

Knapsack Example: Backtracking

(2,2,3;11) means that two elements of each weight 2 and one element of weight 3 is with total value 11



Algorithm

Knapsack Algorithm: Backtracking

BackTrack(i, r) _\\ BackTrack(1, C)

b ← 0

{try each kind of item in tern}

for k ← i **to** n

do

if w(k) ≤ r **then**

b ← max (b, v[k] + BackTrack(k, r - w[k]))

return b

Queens Problem