Logic Programming

Rodica Potolea
Camelia Lemnaru

Lecture #13,
CS@TUCN

Agenda

- Hamiltonian cycle – somehow NP complete/hard problems can be addressed
- Answering your questions & typical errors (this and next lecture)

Hamiltonian cycle with branch and boud

- Branch and bound – oral review
- Estimate the cost of the path in 2 steps, as follows:
  - $\varphi i=\psi i+Xi$
  - φi=cost of the path from Start to Stop via the intermediate node
  - ψI = cost of the path from Start to the intermediate node
  - Xi = cost of the path from the intermediate node to Stop
  - They are estimated: $E\varphi i=E\psi i+Exi$
  - $E\varphi i$=estimate cost of the path from Start to Stop via the intermediate node
  - $E\psi I$ = estimate cost of the path from Start to the intermediate node
  - EXi = estimate cost of the path from the intermediate node to Stop

  - $E\psi I= \psi I$
  - Exi = the better the estimate, more we narrow the search space (bound more effective). Even a rough estimate cold help. Even just 0 proves good.
- The approach (similar bf_serch) is also using 2 argument lists:
  - Candidate list – potential path to follow (it is a regular list; not incomplete)
  - Expanded list – part of paths already covered (from start to current node; none complete to objective)

Hamiltonian cycle with branch and boud - contd

- An element in the Cand list looks like: [n(X,Fi,Length)|Lx]
  - X=node explored
  - Fi=estimate value of φi Eφi=Eψi+Exi = ψi+0
  - Length = number of nodes so far, from Start cu current (X)
  - Lx = parent node = actually the whole path from parent to Start, backwards. It is a list!
- Helping predicates:

```
eqlength(Lg, [n(_,_,Lg)|_]).//Accesses the length of an element in the Candidate list
eqn(X,n(X,_,_)).                    //Accessing the node
ord([n(_,Xfi,_)|_],[n(_,Yfi,_)|_]):-
      Xfi<Yfi.                      //Compares the EψI functions for nodes X and Y
member_thread(X,[H|_]):-
      eqn(X,H).                     //checks if a node is in a list
member_thread(X,[_|T]):-
      member_thread(X,T).
ins_ord_list(X,[H|T],[X,H|T]):-ord(X,H),!. //adds an element
ins_ord_list(X,[H|T],[H|R]):-//in the right (ordered by Fi function) place in the list
      ins_ord_list(X, T,R).
ins_ord_list(X,[],[X]).    //REWRITE the insert predicate with just 2 clauses
```

Hamiltonian cycle with branch and bound - contd

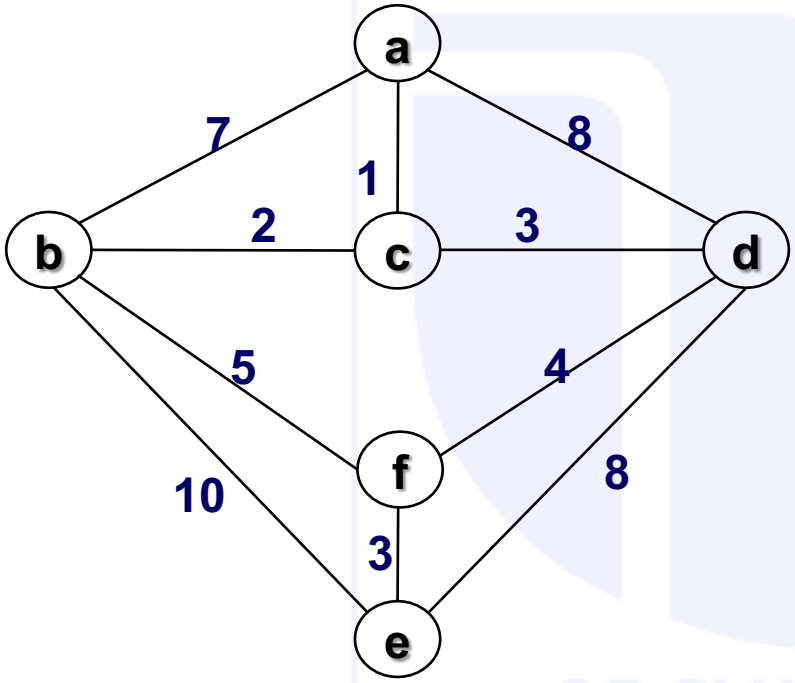
```
hamilton(N,X,Way):-
        search(N,X,Way,[[n(X,0,0)]],[]).
```
//searches for N nodes to end in X the way; put in Cand a path containing just one element: `[n(X,0,0)]` `start` node `X`, estimated weight of ham `EXi=0`, number of nodes `0`, NO parent. Expanded list empty.

```
search(N,X,Way,_,[Way|_]):-  //stop with path the way in front of Expanded
        eqlength(N,Way),!.  //if its length equals the number of nodes in graph
search(N,X,Way,[Y|Cand],Exp):-
        expand(N,X,Y,Cand,NewCand),//if not, expand best node so far=in the element in front of current Cand
        search(N,X,Way,NewCand,[Y|Exp]). //and continue after expansion with the new Cand

expand(N,X,[n(Y,Fi,Length)|Ly],_,_):-
        is_edge(Y,Z,W),  //nondeterministically take Z,  first neighbor of Y (eventually all of them) and weight W
        (not(member(Z,Ly));(Length is N-1,Z=X)), //should NOT be already processed just if is the source
        FiW is Fi+W,Length1 is Length+1,    //estimate the new parameters
        assertz(desc(n(Z,FiW,Length1))),    //put it in the akb
        fail.                               //backtrack to a new neighbor of Y
expand(_,_,L,Cand,NewCand):-
        assertz(desc(end)),                 //mark the end of akb
        collect(L,Cand,NewCand).            //start collecting from the akb and place in NewCand
```

Hamiltonian cycle with branch and boud - contd

```
collect(L,Cand,NewCand):- //L is the parent, a whole path. Is Y in Candidate from clause 2 in search
      get_next(Z),!,                          //take top from akb
      ins_ord_list([Z|L],Cand,IntCand),//add in Cand with the whole path = with L=parent of path
      collect(L,IntCand,NewCand).     //continue with the Intermediate Candidate
collect(_,Cand,Cand).

get_next(Z):-
      retract(desc(Z)),!,
      Z \= end.
```

Example



Cand – ***ordered*** *list of candidate partial paths, waiting expansion*

Exp – *accumulator list of expanded partial paths (also ordered)*

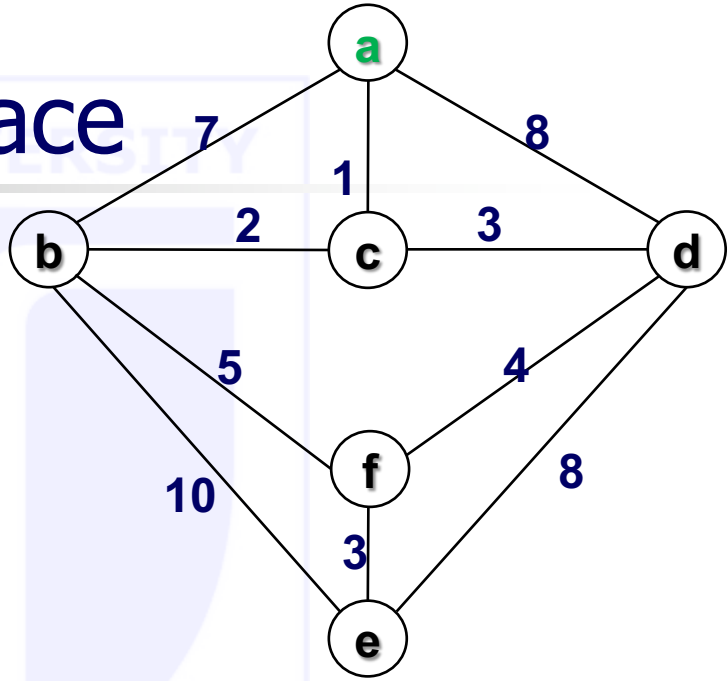
```
?- search(6,a,Way,[[n(a,0,0)]],[]).
```

result list; at the end (crt length = N) it is first element of ***Exp***

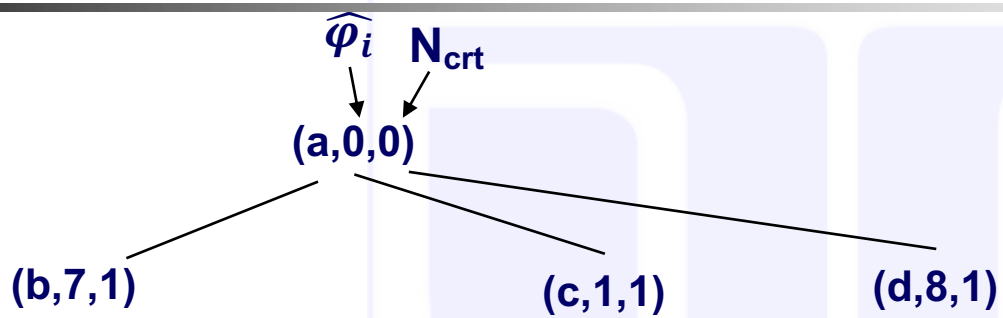
Example – search space

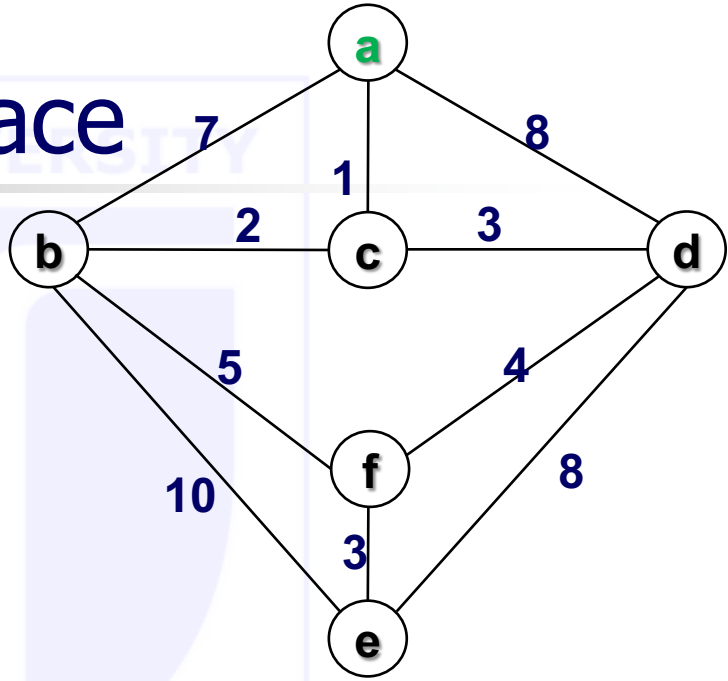
$\widehat{\varphi_i}$ N_{crt}

(a,0,0)

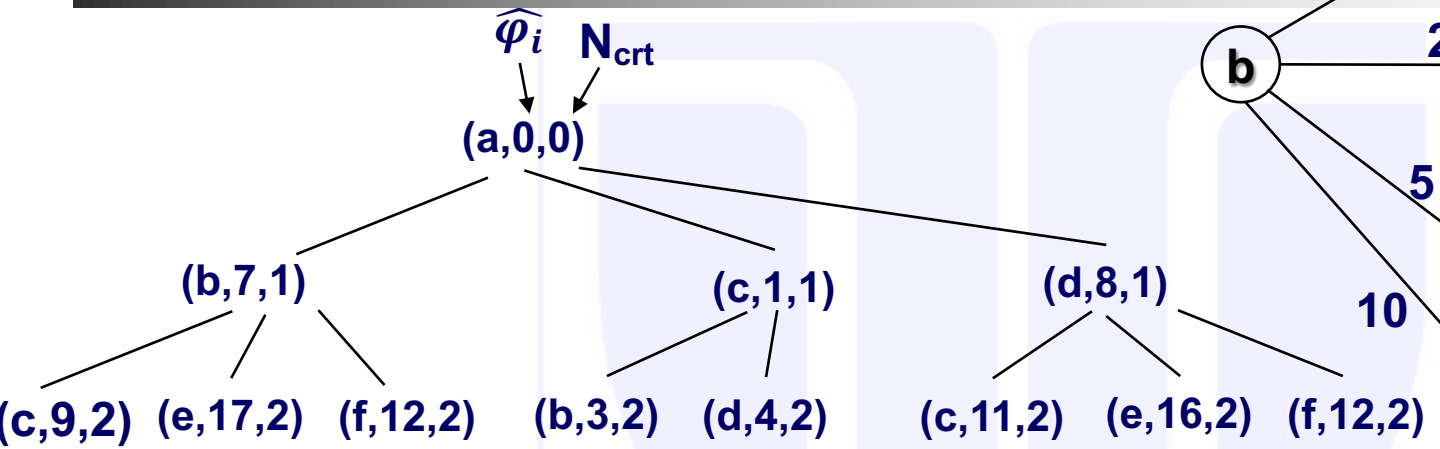


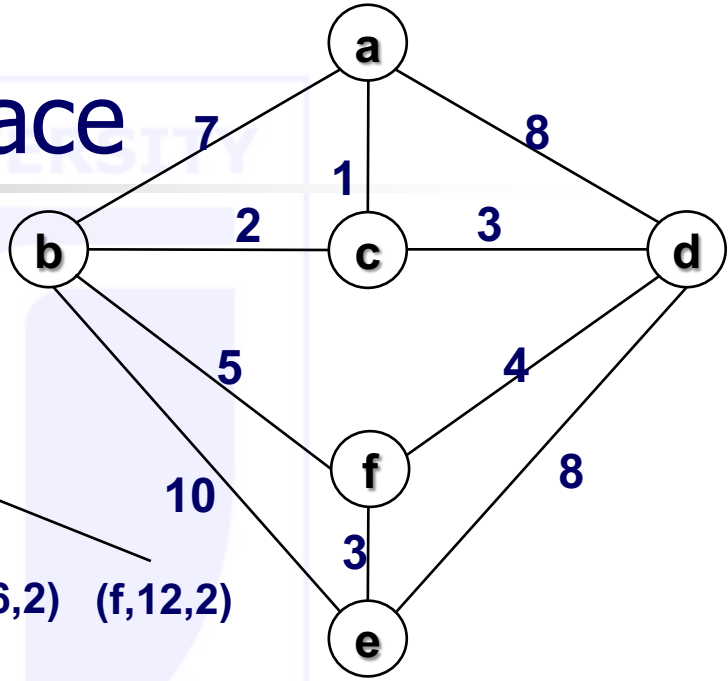
Example – search space



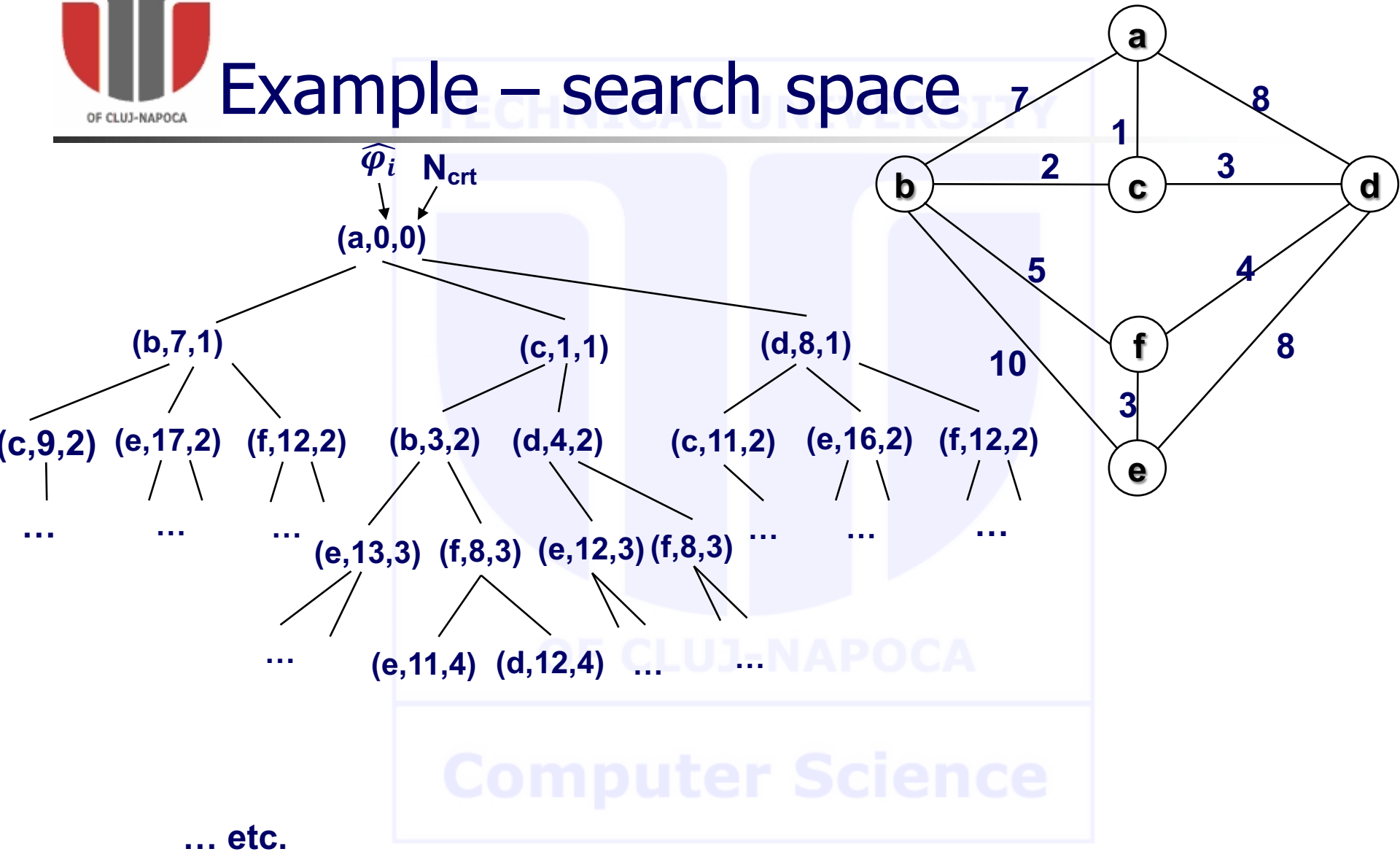


Example – search space



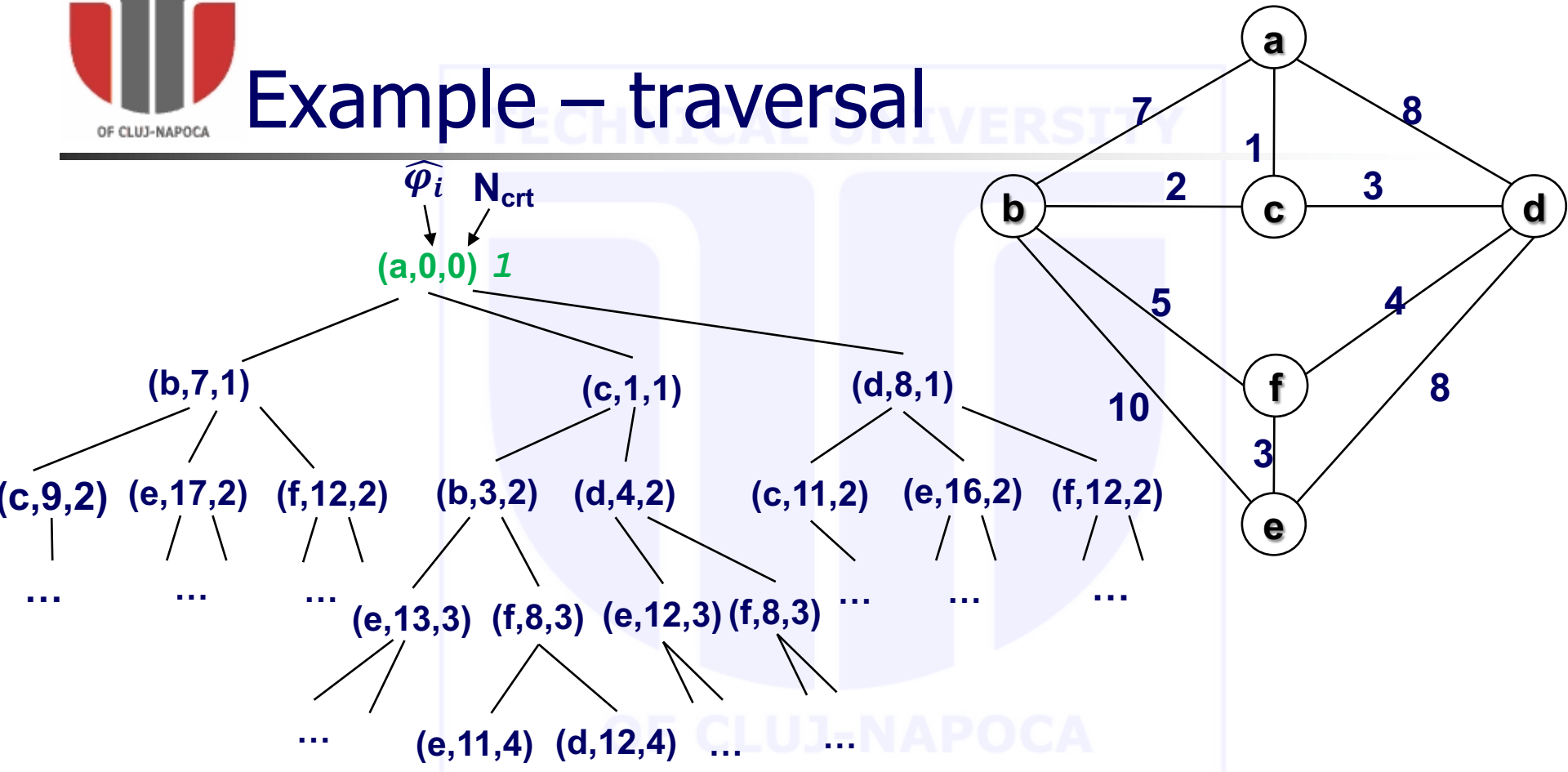


Example – search space



… etc.

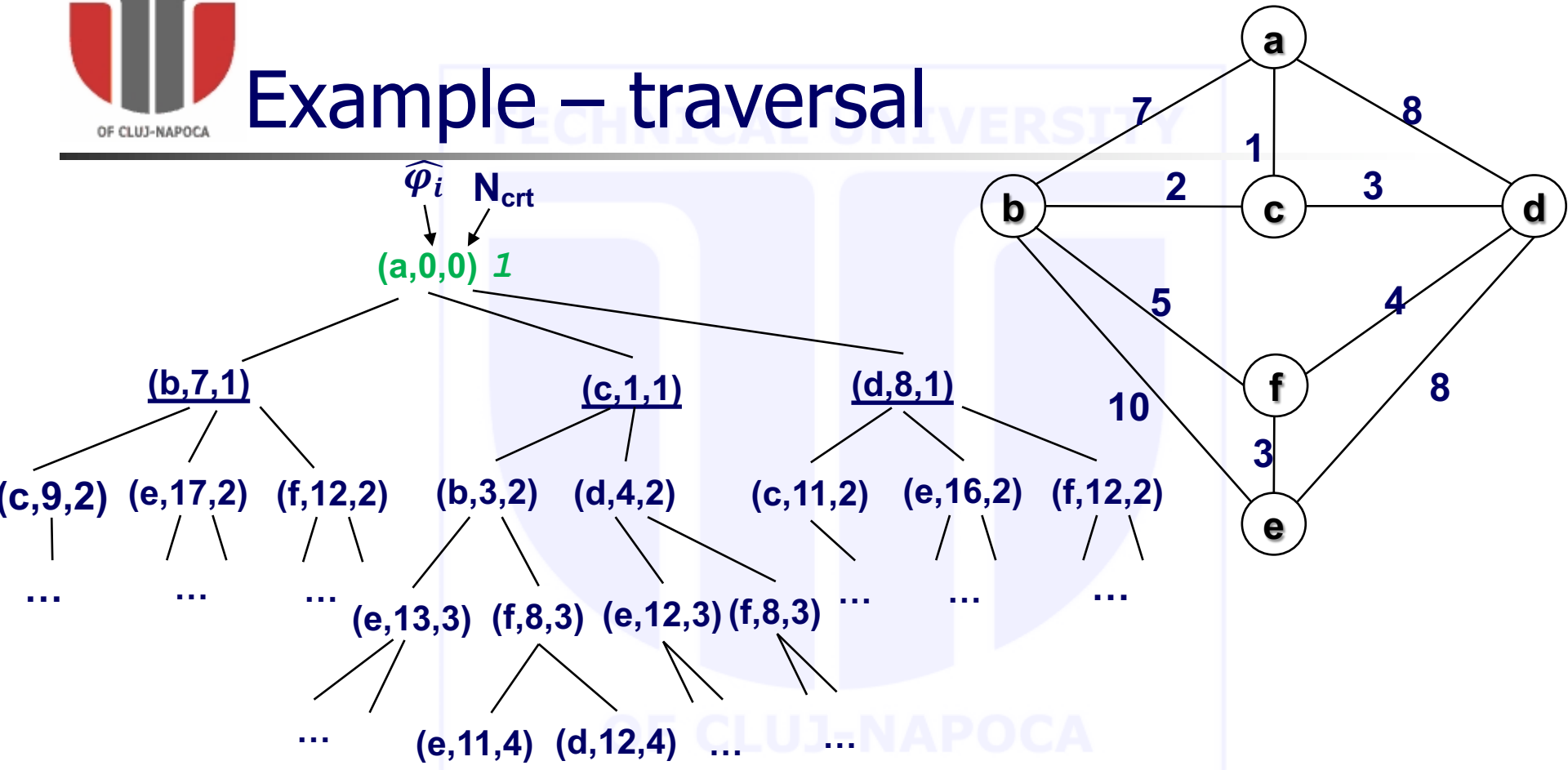
Example – traversal




```
Cand   = [[n(a,0,0)]]
Exp    = []
```

… expand first element in `Cand`

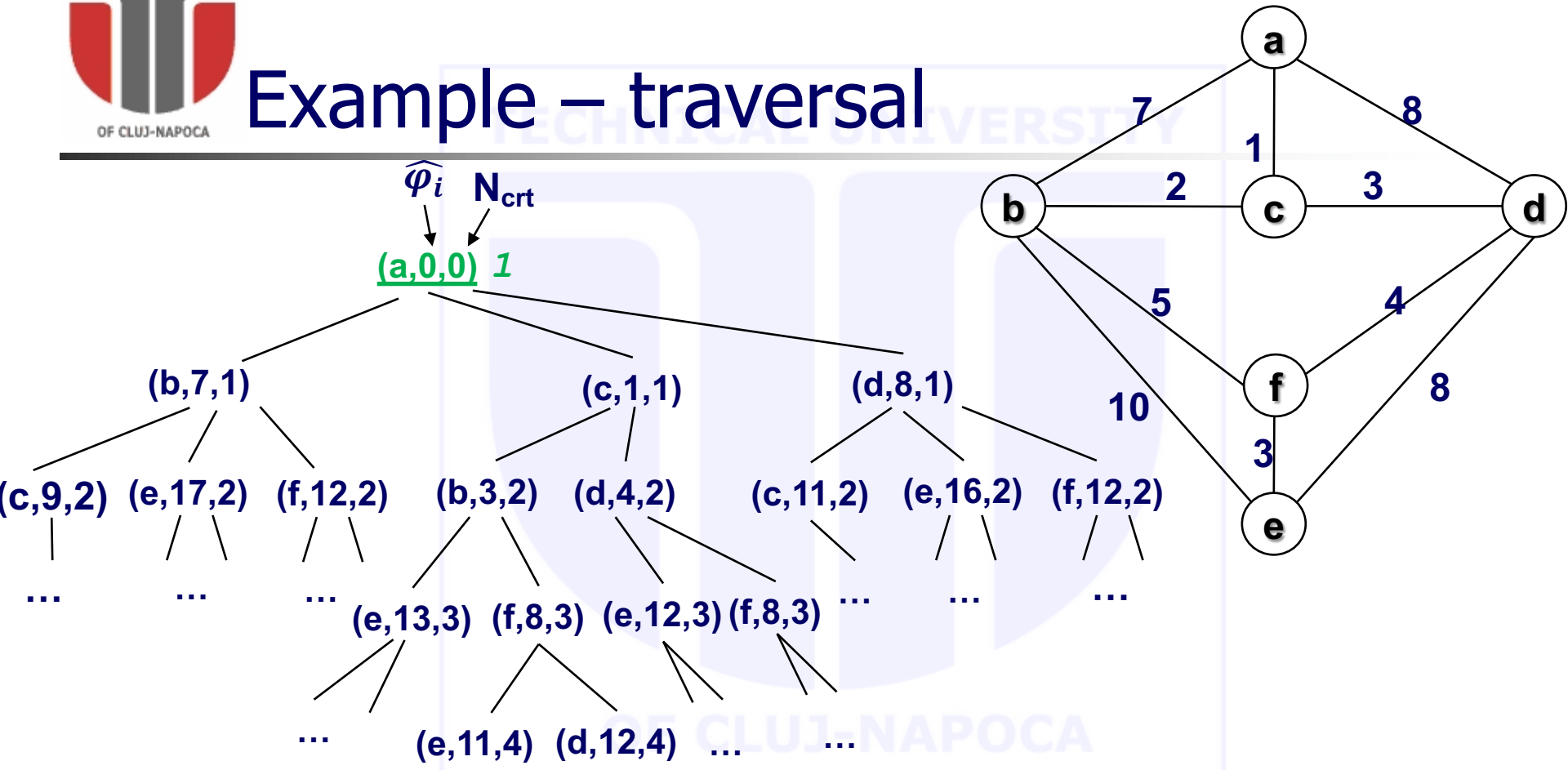
Example – traversal



… after expanding `[n(a,0,0)]`

```
Cand   = [[n(c,1,1),n(a,0,0)], [n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)]]
Exp    = []
```

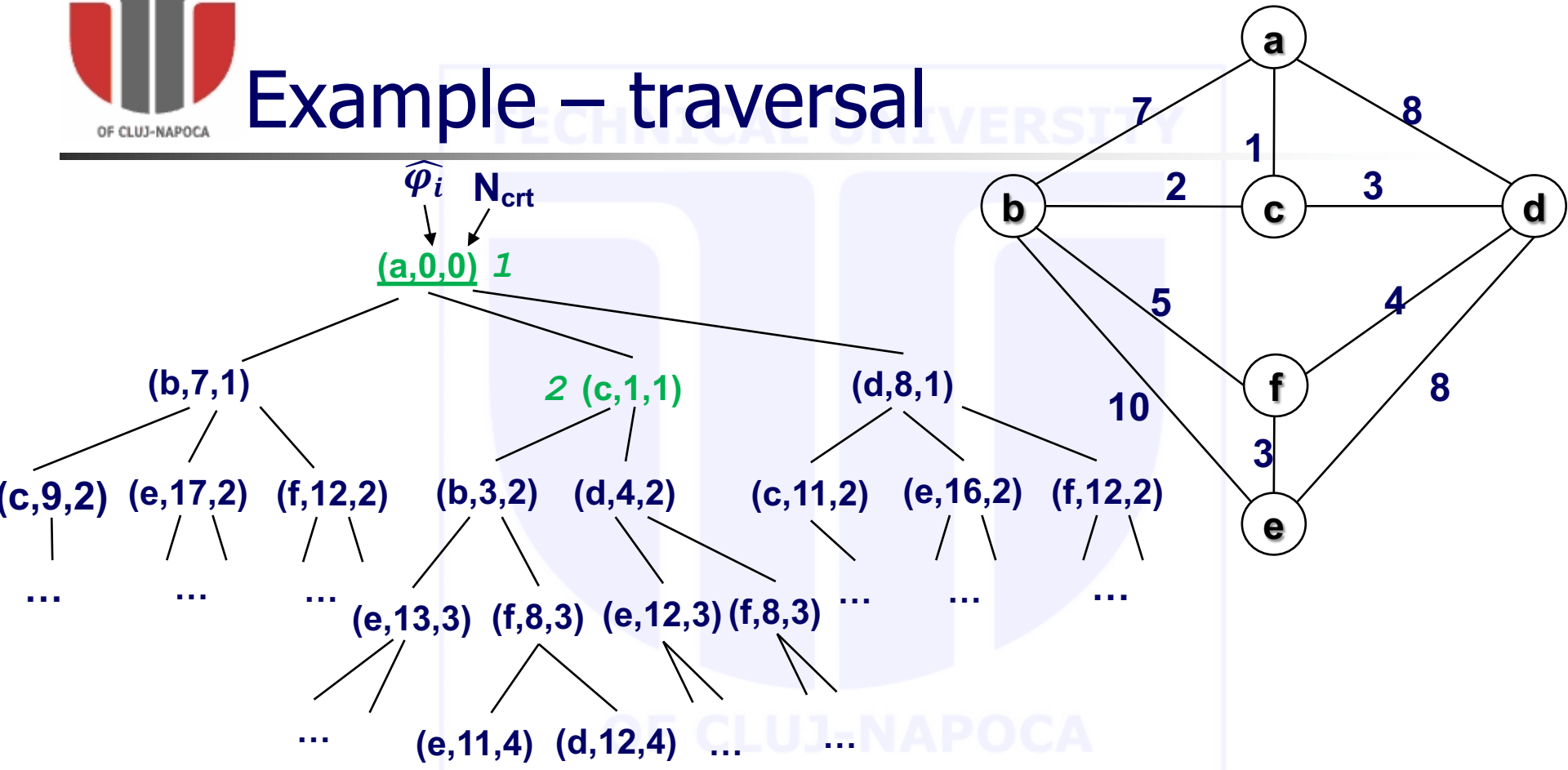
Example – traversal



… recursive call search

```
Cand   = [[n(c,1,1),n(a,0,0)], [n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)]]
Exp    = [[n(a,0,0)]]
```

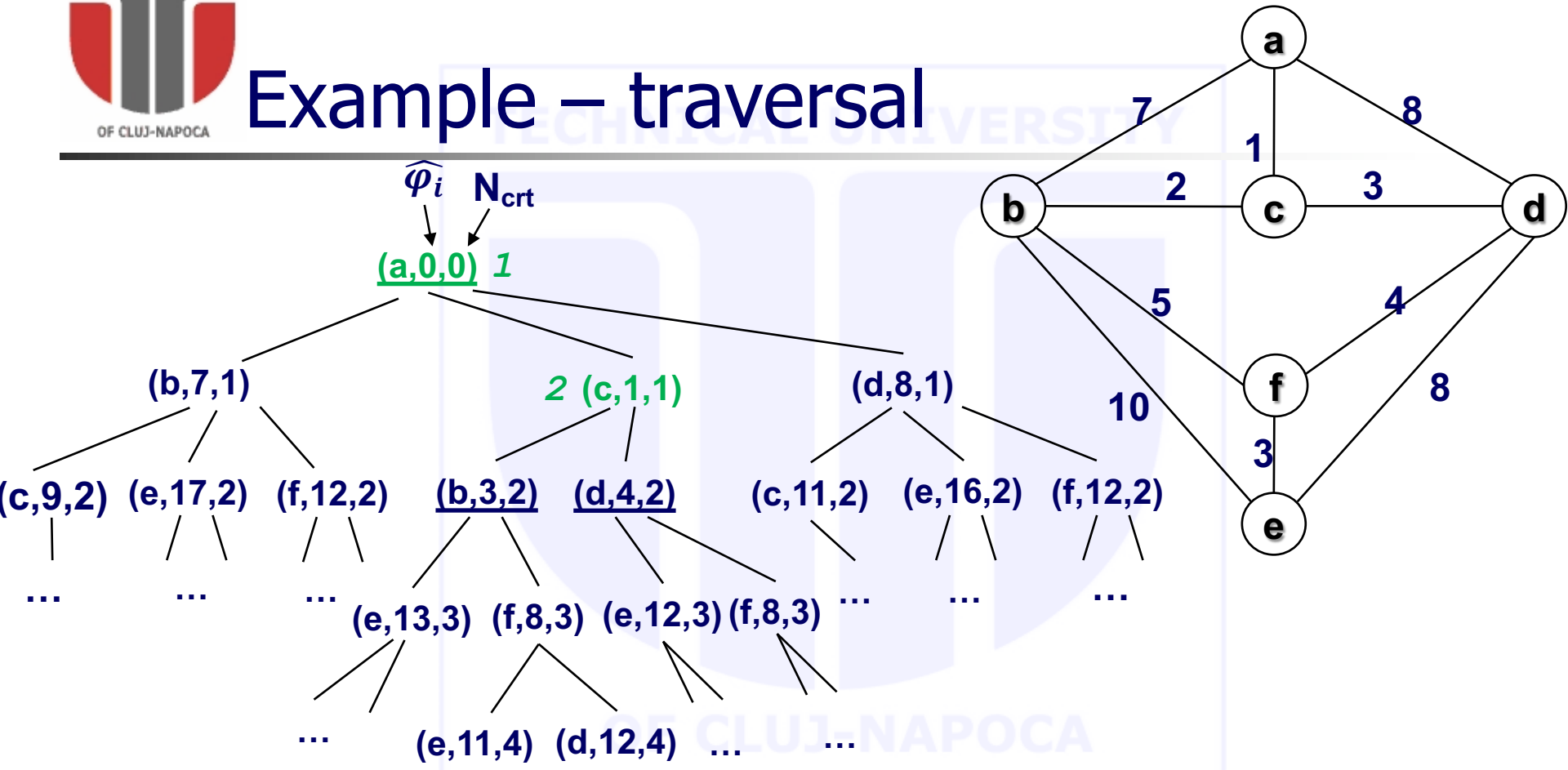
Example – traversal



… this will be expanded next

```
Cand   = [[n(c,1,1),n(a,0,0)], [n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)]]
Exp    = [[n(a,0,0)]]
```

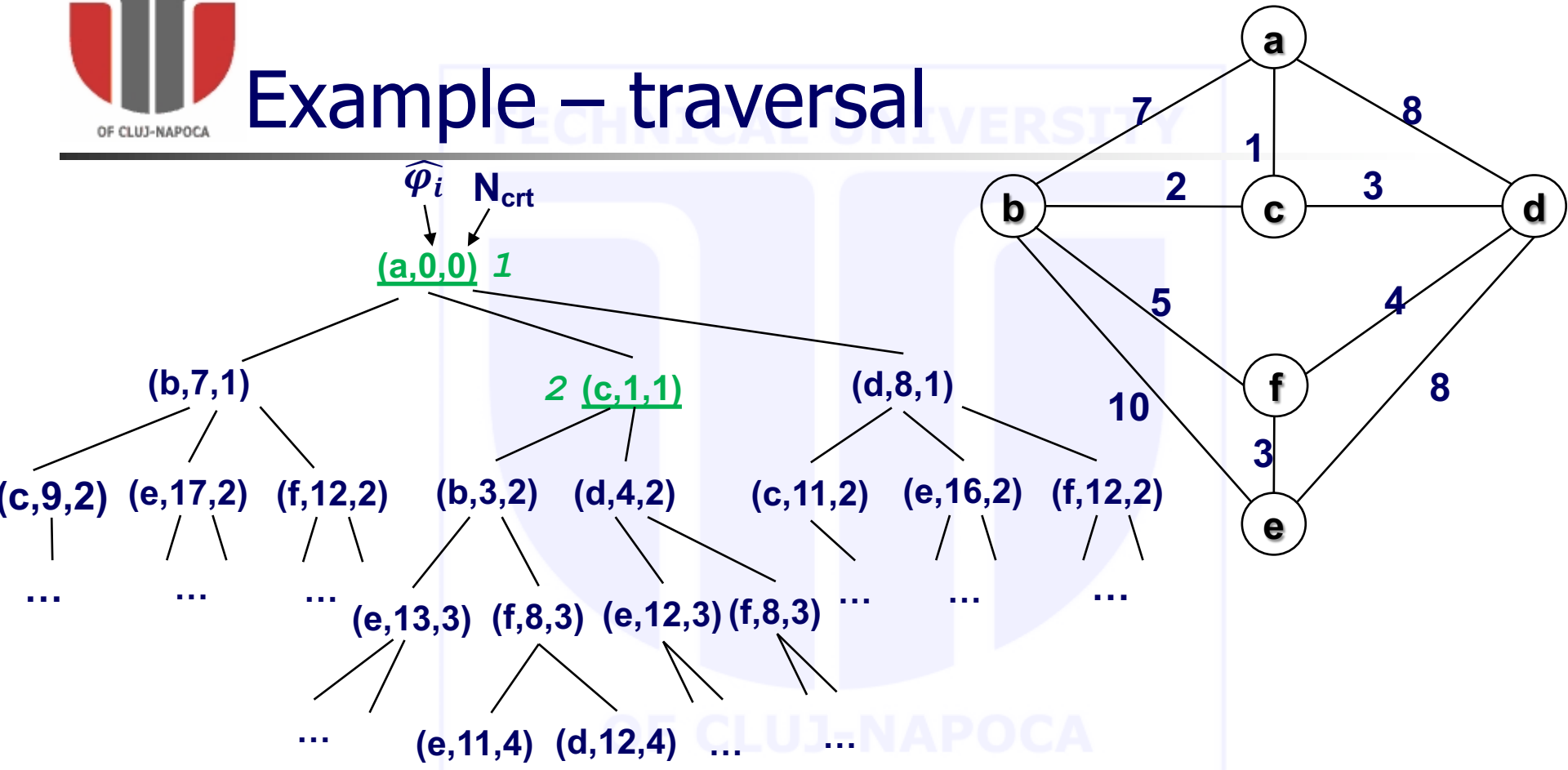
Example – traversal



… after expanding `[n(c,1,1),n(a,0,0)]`

```
Cand   = [[n(b,3,2),n(c,1,1),n(a,0,0)],
          [n(d,4,2),n(c,1,1),n(a,0,0)],[n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)]]
Exp    = [[n(a,0,0)]]
```

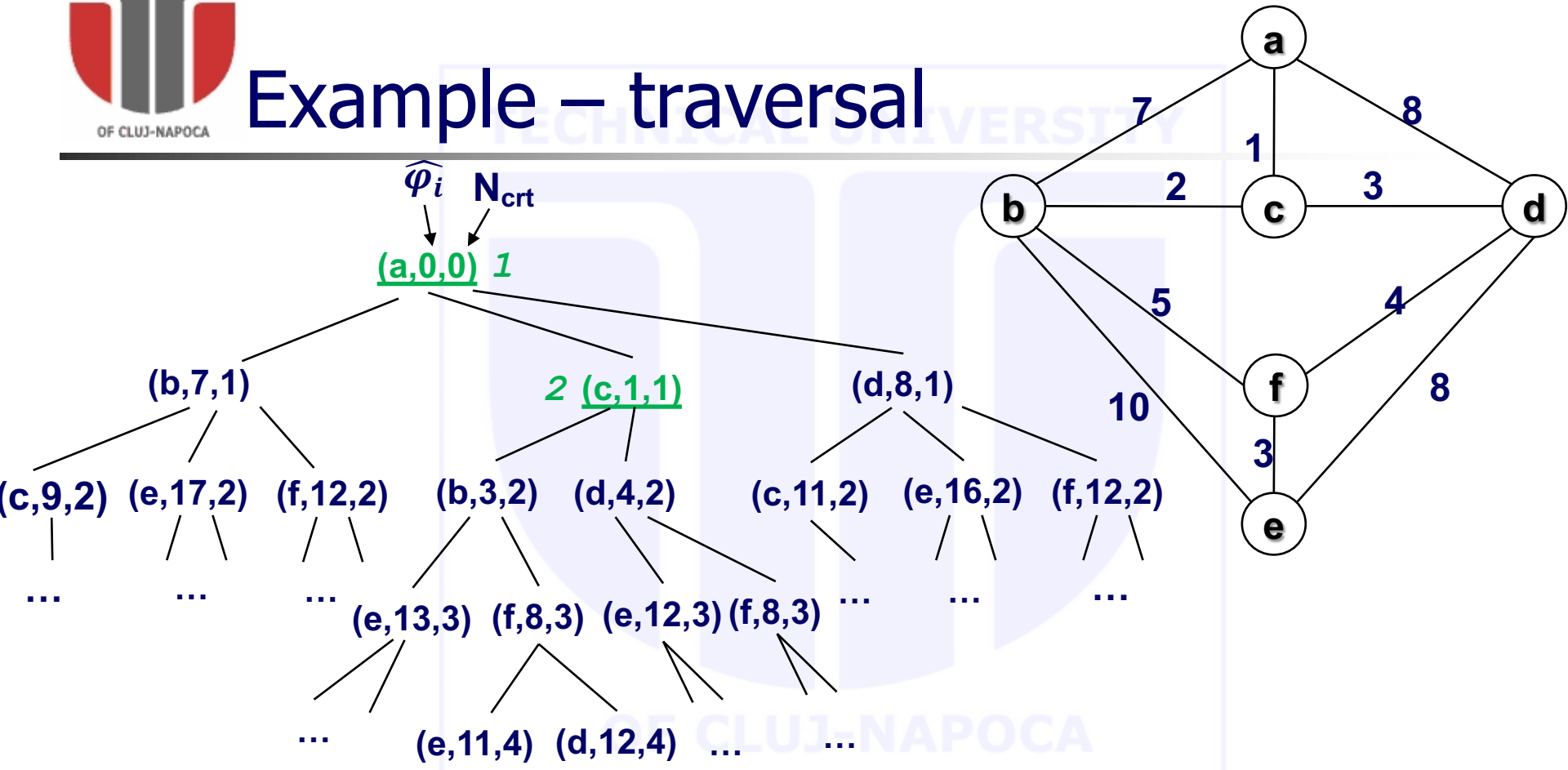
Example – traversal



… recursive call search

```
Cand   = [[n(b,3,2),n(c,1,1),n(a,0,0)],
          [n(d,4,2),n(c,1,1),n(a,0,0)],[n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)]]
Exp    = [[n(c,1,1),n(a,0,0)], [n(a,0,0)]]
```

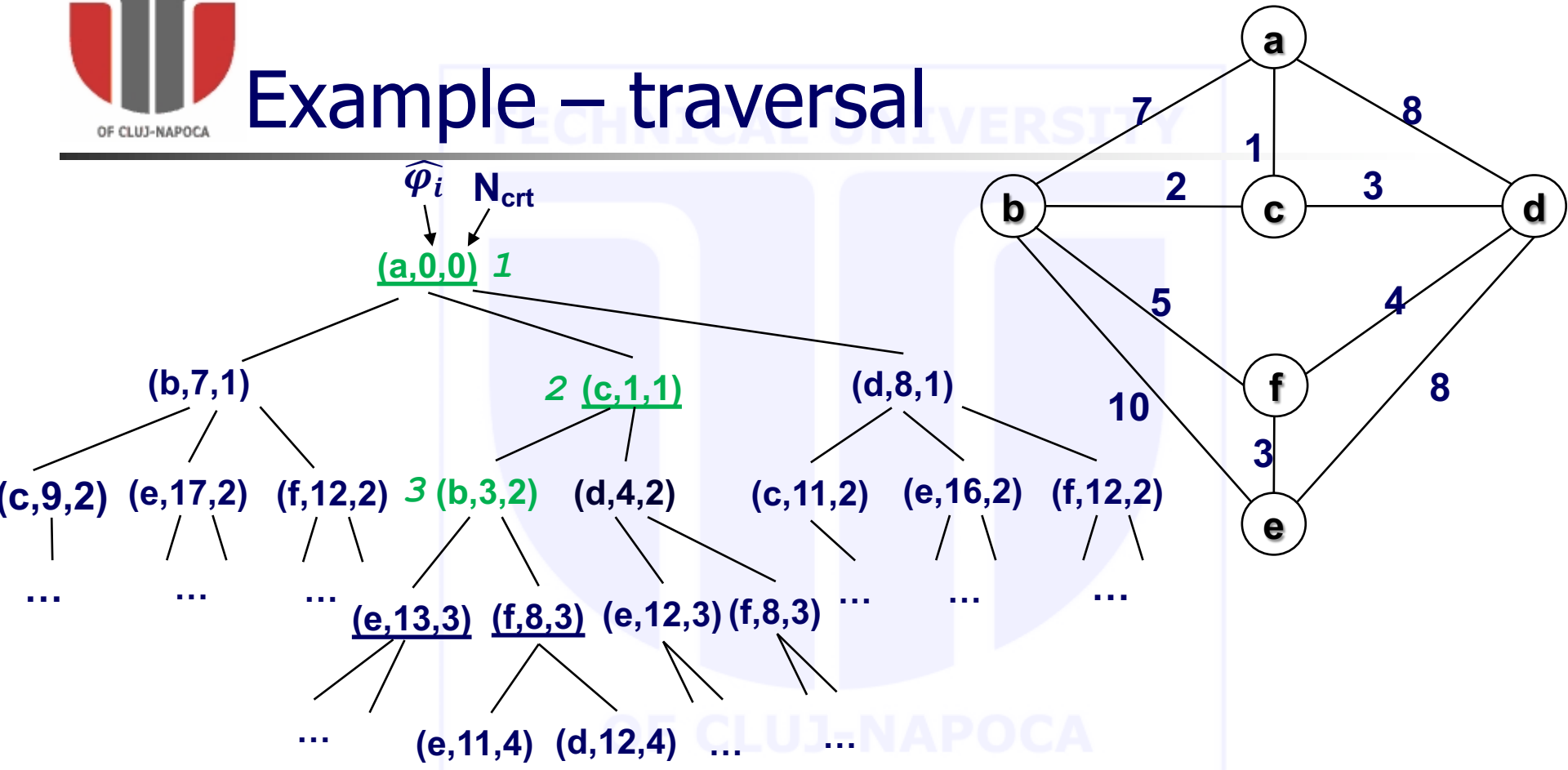
Example – traversal



… this will be expanded next

```
Cand   = [[n(b,3,2),n(c,1,1),n(a,0,0)],
          [n(d,4,2),n(c,1,1),n(a,0,0)],[n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)]]
Exp    = [[n(c,1,1),n(a,0,0)], [n(a,0,0)]]
```

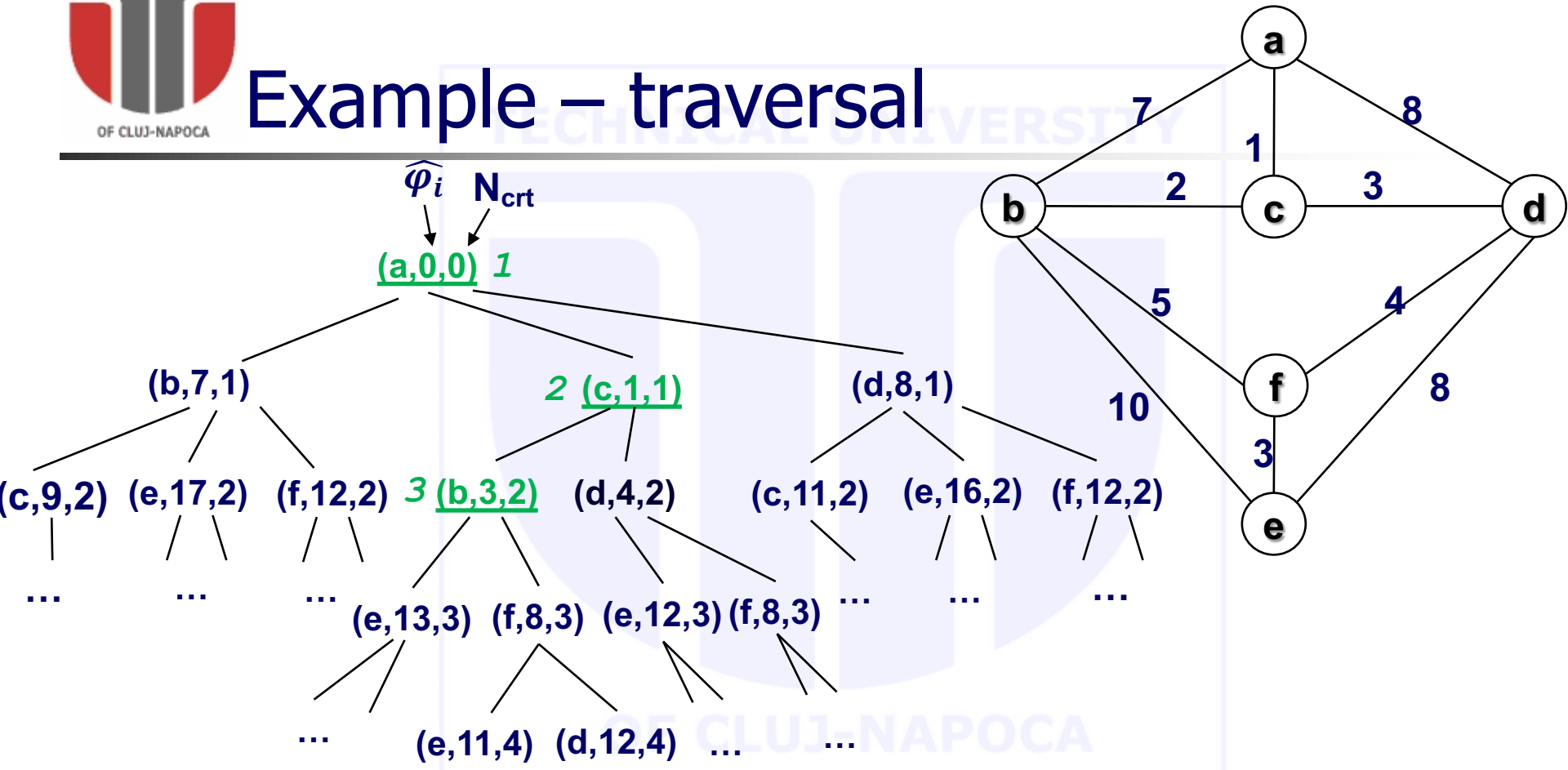
Example – traversal



… after expanding `[n(b,3,2),n(c,1,1),n(a,0,0)]`

```
Cand   = [[n(d,4,2),n(c,1,1),n(a,0,0)], [n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)],[n(f,8,3),n(b,3,2),n(c,1,1),n(a,0,0)],
          [n(e,13,3),n(b,3,2),n(c,1,1),n(a,0,0)]]
Exp    = [[n(c,1,1),n(a,0,0)], [n(a,0,0)]]
```

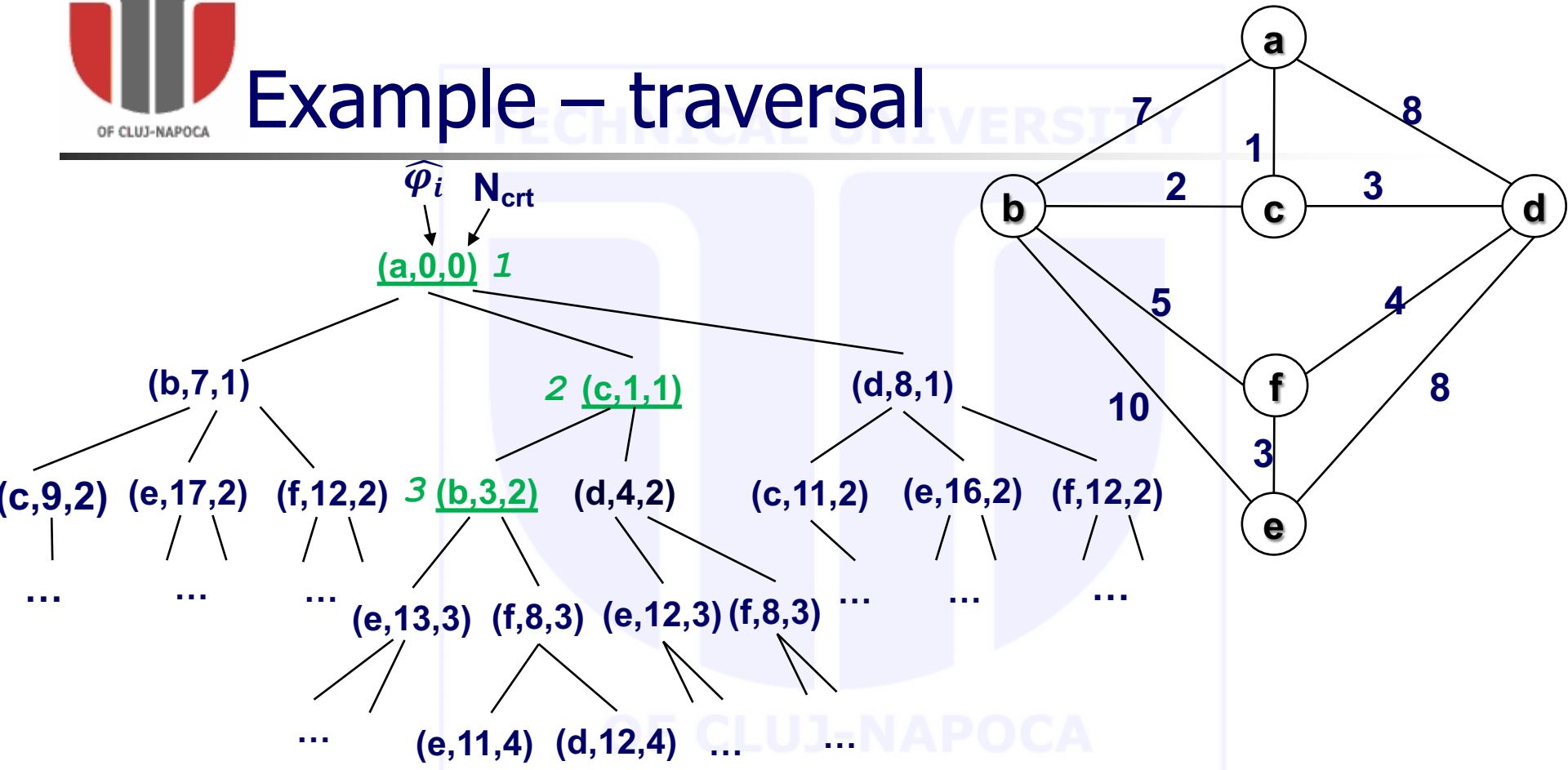
Example – traversal



… recursive call search

```
Cand   = [[n(d,4,2),n(c,1,1),n(a,0,0)], [n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)],[n(f,8,3),n(b,3,2),n(c,1,1),n(a,0,0)],
          [n(e,13,3),n(b,3,2),n(c,1,1),n(a,0,0)]]
Exp    = [[n(b,3,2),n(c,1,1),n(a,0,0)], [n(c,1,1),n(a,0,0)],
          [n(a,0,0)]]
```

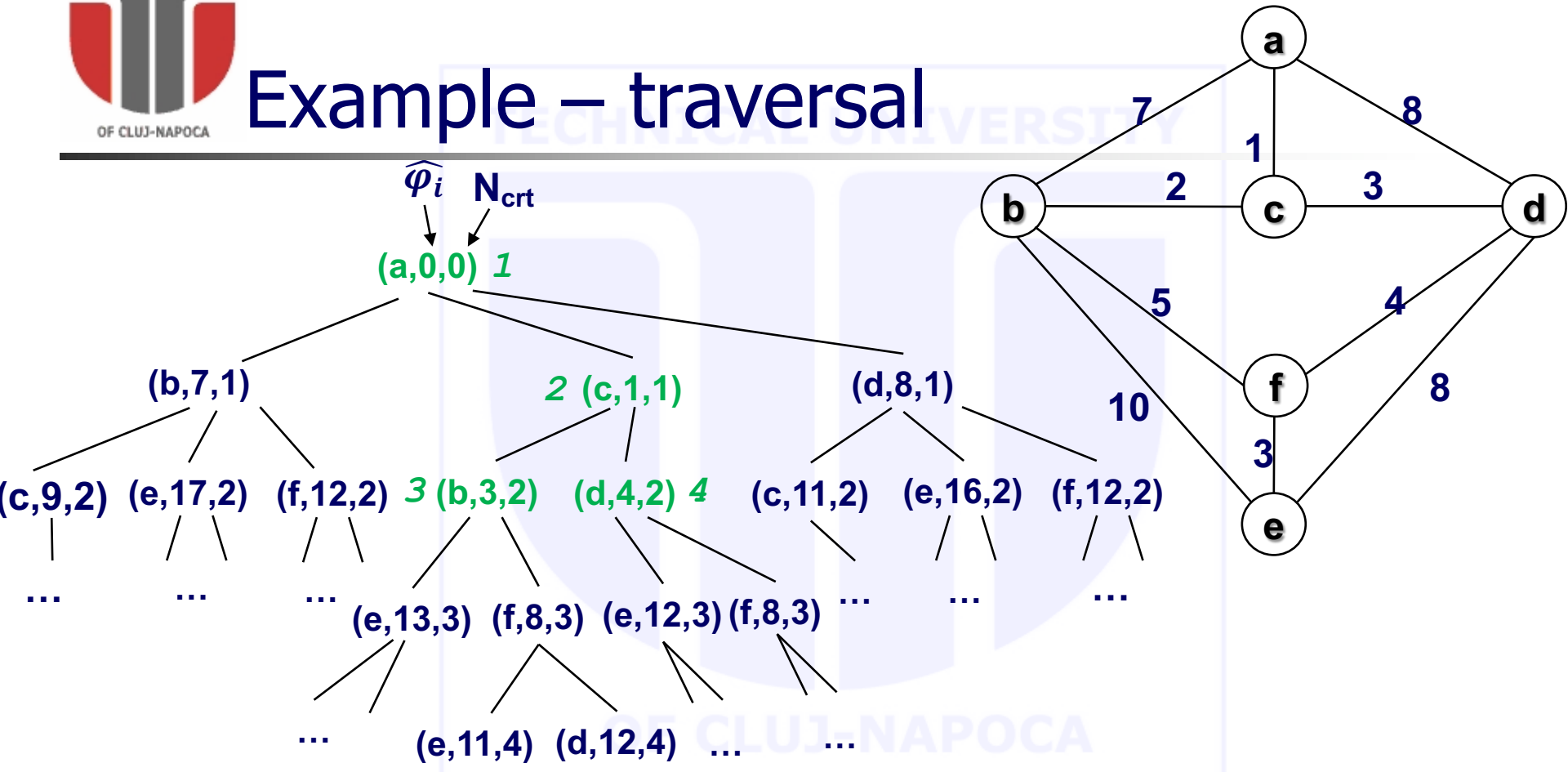
Example – traversal



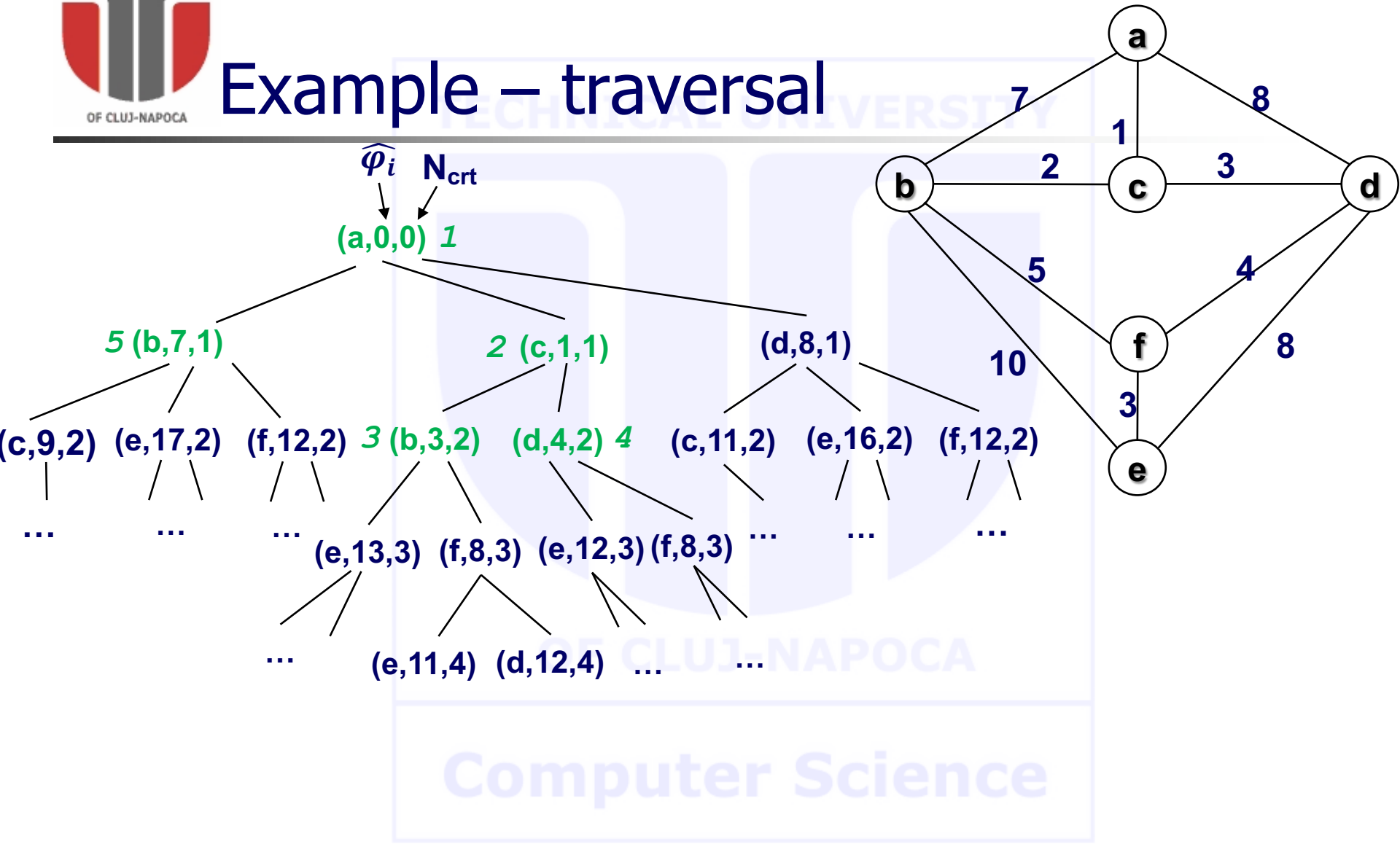
… this will be expanded next

```
Cand   = [[n(d,4,2),n(c,1,1),n(a,0,0)], [n(b,7,1),n(a,0,0)],
          [n(d,8,1),n(a,0,0)],[n(f,8,3),n(b,3,2),n(c,1,1),n(a,0,0)],
          [n(e,13,3),n(b,3,2),n(c,1,1),n(a,0,0)]]
Exp    = [[n(b,3,2),n(c,1,1),n(a,0,0)], [n(c,1,1),n(a,0,0)],
          [n(a,0,0)]]
```

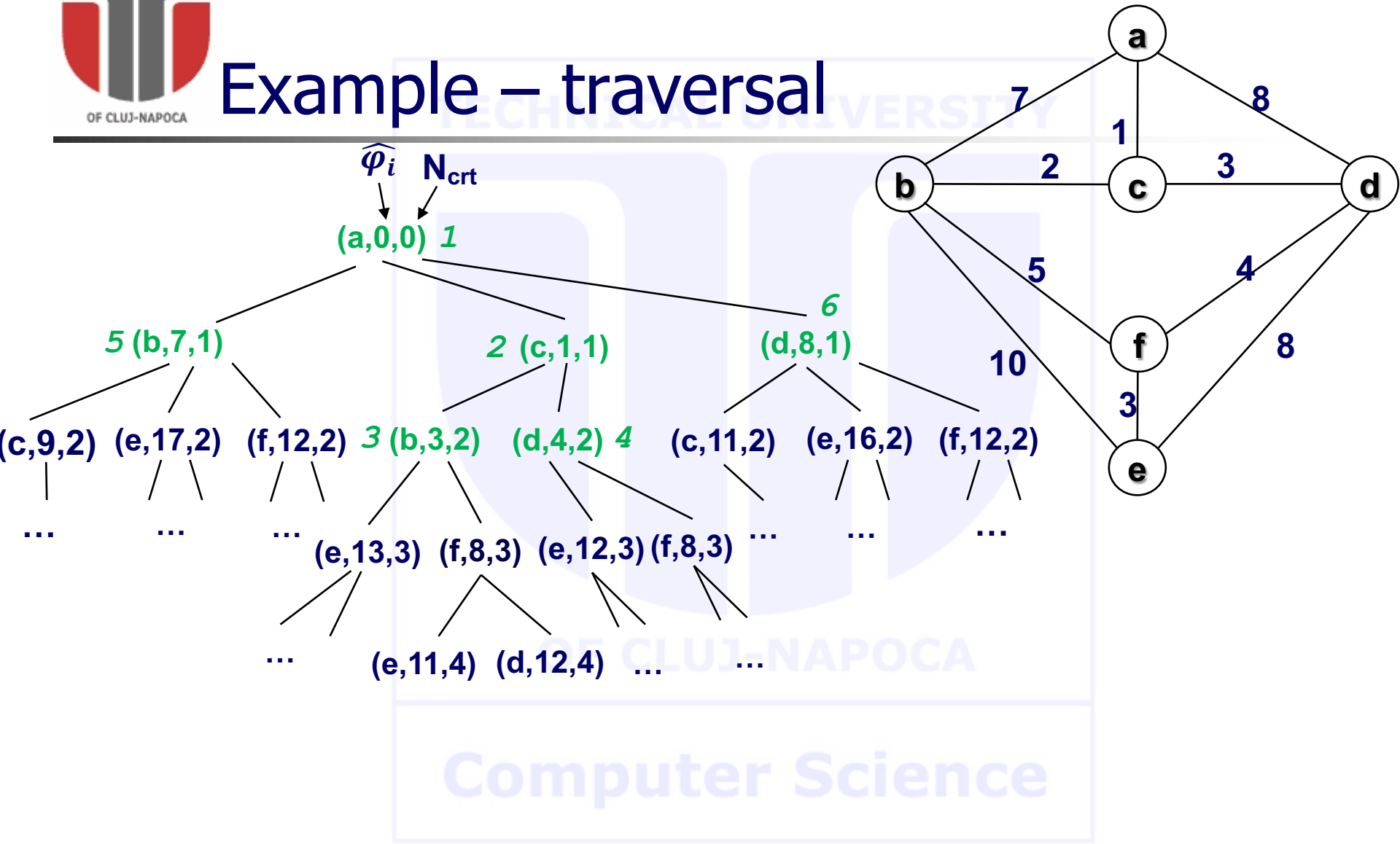
Example – traversal



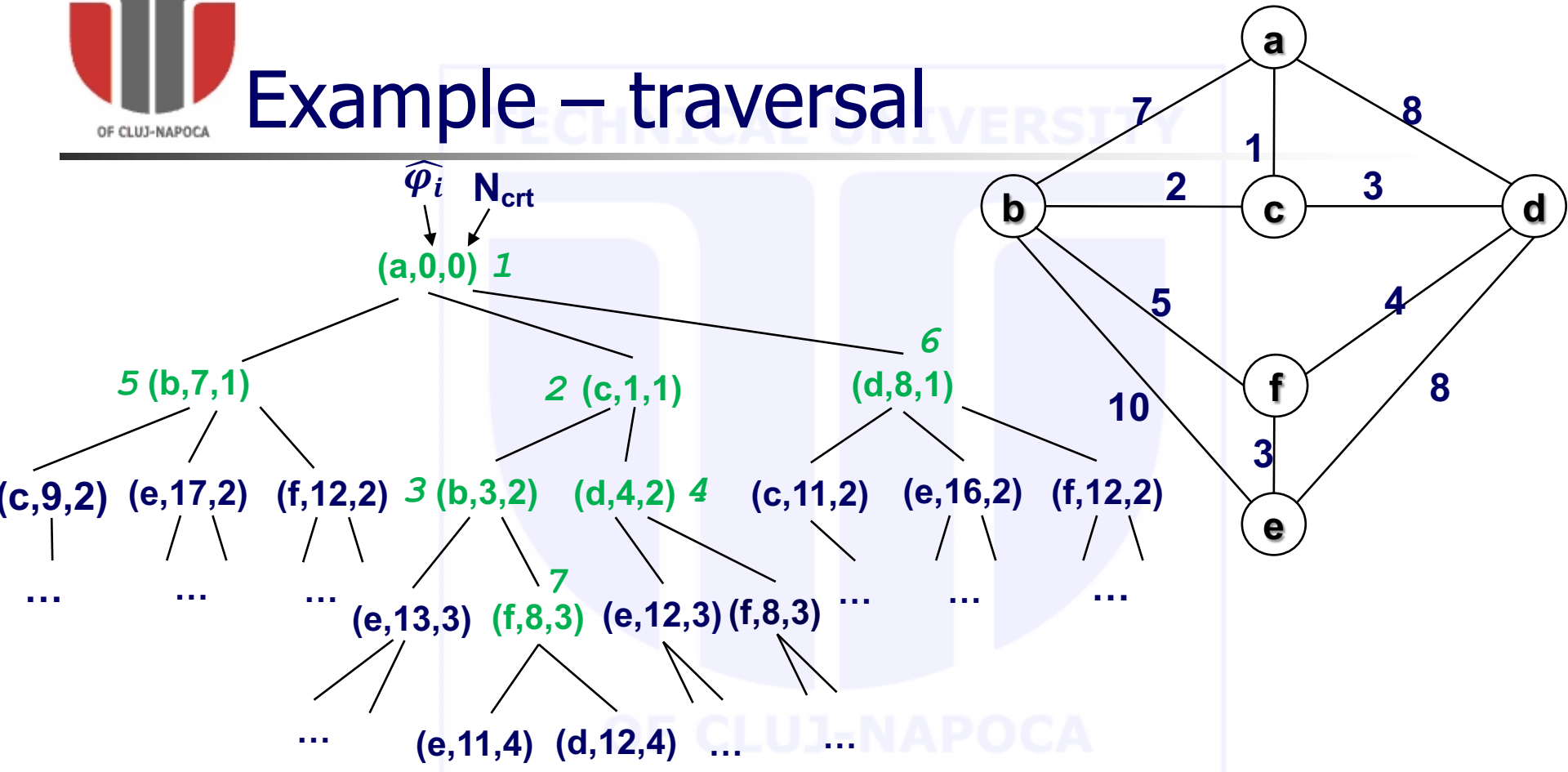
Example – traversal



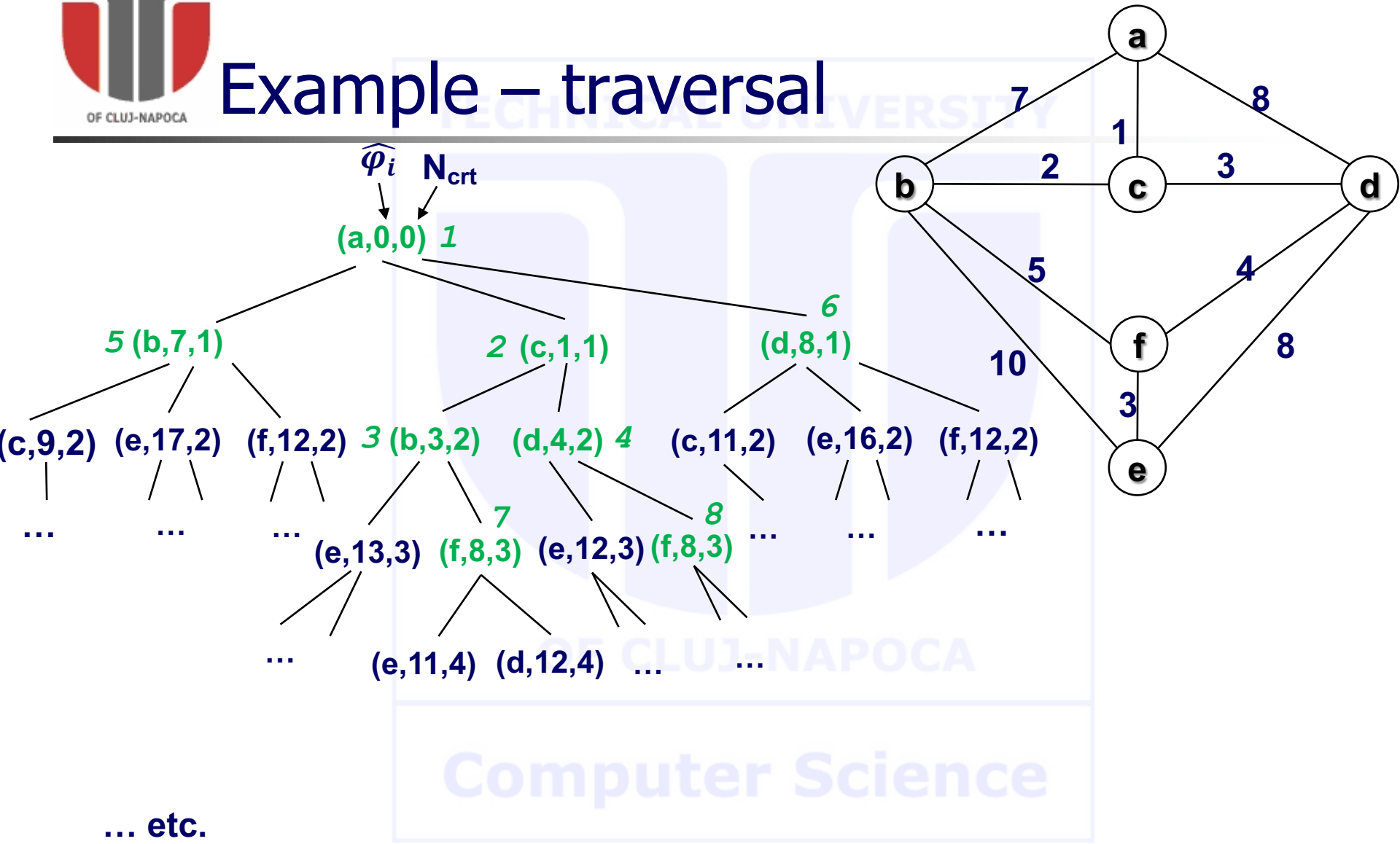
Example – traversal



Example – traversal



Example – traversal



… etc.