3. SPECIFICAȚII DE PROIECTARE

3.1. Preliminarii

Limbajul utilizat pentru dezvoltarea Sistemului Expert **PManager** este C++. Mediul de programare în care s-a implementat sistemul este Microsoft Visual Studio, versiunea 6.0.

Visual C++ 6.0 este ultima şi cea mai bună versiune a compilatorului Microsoft Visual C/C++. Produsul a devenit mult mai mult decât un simplu compilator, având incluse clase fundamentale Microsoft, care simplifică şi accelerează dezvoltarea aplicațiilor Windows, editoare sofisticate de resurse în scopul proiectării casetelor de dialog complexe, a meniurilor, a barelor cu instrumente, imaginilor şi a multor alte elemente ce compun aplicațiile Windows moderne. Este oferit un excelent mediu de dezvoltare integrat, numit Developer Studio, care furnizează forme grafice pentru structurile aplicației pe măsură ce se dezvoltă, un instrument pentru depanare integrat, care permite inspectarea în detaliu al fiecărui aspect din cadrul unui program aflat în execuție. Acestea sunt doar câteva din numărul mare de facilități oferite de Visual C++ 6.0, în care se dezvoltă aplicații rapide şi complete, folosind cele mai recente tehnologii din Windows.

```
id22
45
30
neantelegerea corecta a cerintelor clientilor
id31
120
60
refacerea interfetei dintre clase
id14
45
60
erori de sincronizare
id37
300
60
erori de sincronizare
id39
120
60
reorganizare pe clase
```

3.2. Structuri de date utilizate

Sistemul Expert **PManager** este conceput să genereze trei fişiere care au aceeaşi denumire cu proiectul pe care-l supervizează.

- Primul fişier are extensia "**.pmd**". În acest fişier se găsesc activitățile din care este alcătuit proiectul, structura rețelei proiectului (nodurile sursă şi destinație pentru fiecare activitate), unitatea de timp de planificare, data de început a proiectului, timpul estimat de finalizare şi programul de lucru. Astfel, pentru deschiderea fişierului aferent unui proiect, datele sunt încărcate din fişierul cu extensia ".pmd", având denumirea atribuită de către utilizator. Fişierul ".pmd" este în fapt o arhivă.

- Al doilea fişier are extensia "**.dat**". El conține toate activitățile din cadrul proiectului, cărora li s-a făcut transfer de timp din "*bufferul de timp al proiectului*", durata optimă estimată inițial, durata timpului transferat şi motivele care au generat întârzierea activității. Acest fişier este de fapt modulul "Experiență" al sistemului "**PManager**" Se stochează datele sub formă de cadre, Fig.3.1. Fiecare cadru conține:

Cadru: codul activității

Valoarea:

Fațeta: durata optimă

Valoarea:

Fațeta: timpul transferat

Valoarea:

Fațeta: motivele care au generat transferul

Valoarea:

Fig. 3.1. Fişierul cu extensia ".dat"

- Al treilea fişier are extensia **".crt**". El conține evoluția drumului critic, ținând cont de faptul că transferurile de buffer procesate pe parcurs au generat modificarea repetată a secvenței drumului critic, Fig.3.2. Cadrele acestui fişier au următoarea structură:

Cadru: secvența evenimentelor critice

Valoarea:

Fațeta: activitatea care a beneficiat de transfer de buffer

Valoarea:

Fațeta: valoarea transferului de timp

Valoarea:

Fațeta: valoarea actualizată a drumului critic

Valoarea:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 3
id22
30
4155
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 3
id31
60
4185
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 3
id14
60
4185
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 3
id37
60
4185
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 3
id39
60
4245
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 3
id31
60
4305
```

Fig. 3.2. Fişier cu extensia ".crt"

Toate informațiile referitoare la activitățile unui fişier „.pmd" sunt concepute într-o listă simplu înlănțuită. Dacă la un anumit moment trebuie accesate anumite informații cu privire la o activitate, nu este necesară deschiderea fişierului „.pmd", putându-se parcurge mult mai rapid lista activităților în care se regăsesc toate informațiile respective.

Tipul de dată "POSITION" este o variabilă cu ajutorul căreia se identifică poziția unui element dintr-o colecție (liste), folosind clase care moştenesc liste. În cazul Sistemului **"PManager"** clasa de bază este o de asemenea o listă simplu înlănțuită.

Pe lângă aceste fişiere, structura de date cea mai utilizată este o matrice pătratică "**work"**, având numărul de linii egal cu numărul nodului (destinație) final al rețelei proiectului. Matricea **work,** reprezintă de fapt matricea de adiacență a grafului construit pe baza activităților introduse în proiect. Matricea se inițializează cu "–1", după care se parcurge lista care conține toate activitățile iar matricea - **work**[sursă][destinație] - se complectează cu valoarea drumului între nodul sursă şi nodul destinație. În faza de concepție a sistemului **PManager** s-a dovedit mult mai potrivită matricea de adiacență pentru calculul drumului critic în loc de alternativa utilizării listelor.

O altă structură importantă este matricea - **tab**[61][2] - în care se găseşte factorul "**Z**" (§ 4.2) pe baza căruia se determină probabilitatea de finalizare a proiectului .

În matricea - **t**[numărul de activități][3] - se memorează activitățile finalizate la un moment dat. Pe prima poziție se găseşte nodul sursă al activității finalizate, pe a doua poziție se găseşte nodul destinație, iar pe ultima se găseşte valoarea drumului maxim care se parcurge din nodul de start al proiectului până în nodul sursă al activității respective.

În matricea - **dx**[numărul de activități][4] - se memorează activitățile care sunt în curs de desfăşurare la un moment dat. Pe prima poziție se găseşte nodul sursă al activității care este în curs de desfăşurare, pe a doua poziție se găseşte nodul destinație al activității, pe a treia poziție durata de timp scursă începând cu momentul de start al respectivei activități, iar pe ultima poziție se regăseşte valoarea drumului maxim care se parcurge din nodul de start până în nodul sursă al activității

Vectorul - **v**[număr noduri] - este utilizat pentru a captura drumul maxim de la nodul de start al proiectului şi până la nodul a cărui valoare este dată de indicele vectorului.

Pe lângă structurile de date prezentate mai sus, a fost necesară şi definirea matricii "**cost**" care este similară cu matricea de adiacență "**work**"**,** cu deosebirea că elementele sale returnează costul activității, având pentru nodul sursă valoarea liniei **i,** iar pentru nodul destinație valoarea coloanei **j**.

- Matricea "**costu**" returnează valoarea costului unei unități de timp din cadru unei activități din proiect.
- Matricea "**Costfinal**" returnează valoarea costului unității de timp pentru tot proiectul.
- Matrice "**Coord**[nr de unitati][4]" returnează coordonatele pentru trasarea graficului costurilor.
- **Path** este o listă de valori întregi în care se stochează drumul critic.

3.3. Structura programului

Pentru o abordare cât mai eficientă şi o prezentare expresivă a structurii Sistemului Expert **PManager**, în faza proiectării arhitecturale, sistemul a fost conceput pe baza diagramelor UML

(Unified Modeling Language), fiind considerat un limbaj pentru specificarea, vizualizarea, construirea şi documentarea produselor sistemelor software [Fil-99] [Can-98].(Fig.3.3)

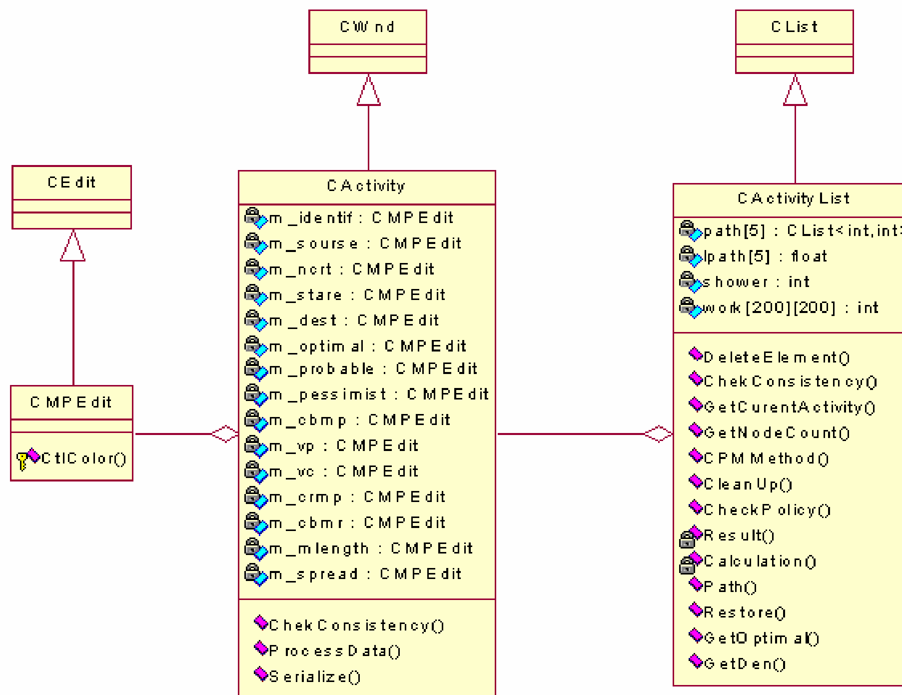


Fig. 3.3. PManager - Detaliu (a) din diagrama UML

Clasele principale sunt prezentate în cele ce urmează:

- **CEdit** - este o clasă definită în *Microsoft Fundation Class Library* şi reprezintă grafic o celulă dreptunghiulară descendenta ferestrei în care se pot introduce texte.
- **CPMEdit -** este o clasă creată în cadrul acestui program, care "moşteneşte" clasa **CEdit**. Clasa **CPMEdit** foloseşte mai multe metode moştenite de la **CEdit** şi *conţine* în plus o metodă *CtlColor()* care setează culoarea. Câteva exemple în acest sens utilizate în program sunt:
  - Metoda *SetWindowText*, care reprezintă o metodă procedurală prin care se scrie într-o căsuţă de editare.
  - Metoda *GetWindowText*, este o metodă procedurală prin care se citeşte ce este scris într-o căsuţă de editare.

- **CWnd -** este o clasă care generează funcţionalităţile de bază pentru toate clasele de tip fereastră în *Microsoft Fundation Class Library.*
- În cadrul programului **PManager**, clasa **CActivity -** este o clasă care *moşteneşte* clasa **CWnd**. Între clasa **CActivity** şi clasa **CPMEdit** s-a stabilit o relaţie de agregare, **CActivity** conţinând 15 obiecte care sunt instanţe ale clasei **CPMEdit**. Pe lângă aceste obiecte, **CActivity** include următoarele metode descrise mai jos:

- *CheckConsistency* - este o metodă care verifică dacă datele aferente unei activități au fost introduse corect. Verifică dacă sursa este mai mică decât destinația, dacă valoarea duratei optime este mai mică decât valoarea duratei probabile, respectiv mai mică decât valoarea duratei pesimiste. În cazul introducerii corecte a datelor, *CheckConsistency* generează un semnal de validare prin intermediul unei valori de tip întreg (int), pentru a valida metoda care urmează.
- *ProcessData* – în cazul semnalului de validare de la procedura precedentă, calculează pentru fiecare activitate timpul mediu, dispersia, variația de cost şi variația de planificare.
- *Serialize* - este o funcție definită în clasa **CObject** (clasă de bază, moştenită la rândul ei de **CWnd**) şi redefinită în cadrul Sistemului **PManager**, care încarcă sau salvează un obiect din / în arhivă.

O activitate din planificator este un exemplu de obiect foarte des creat al clasei **CActivity**.

■ Clasa **CActivityList** este de fapt o listă simplu înlănțuită în care fiecare nod al listei reprezintă o activitate. Clasa **CActivityList** este o clasă care moşteneşte o clasă **CList**, a cărei noduri sunt pointeri către o clasă **CActivity**. Clasa **CList** este o listă definită în *Microsoft Fundation Class Library*.
Între clasa **CActivityList** şi clasa **CActivity** s-a stabilit o relație de agregare. **CActivityList** include metodele procedurale moştenite de la **CList** descrise în cele ce urmează:

- *GetCurrentActivity* - returnează activitatea curentă (ultima activitate).
- *GetHeadPosition* - este o metodă procedurală care nu este implementată în această clasă, dar care este moştenită de la clasa CList. Metoda returnează poziția primului element din listă.
- *GetAt* - este o metodă procedurală moştenită din clasa CList şi returnează activitatea (nodul) care se găseşte pe poziția dată ca parametru.
- *GetNext* - este o metodă procedurală moştenită din clasa de bază CList şi returnează un pointer către activitatea următoare activității care se găseşte pe poziția primită ca parametru.
- *GetNodeCount* - este o metodă procedurală care calculează nodul cel mai mare din listă.
- *GetCount* - este o metodă procedurală moştenită de la clasa de bază **CList** şi returnează numărul de elemente din listă.
- *IsEmpty* - este o metodă procedurală implementată în **CList** şi moştenită de **CActivityList** şi testează dacă lista este goală (nu are elemente). .
- *Serialize* - este o metodă procedurală definită în clasa **CObject** şi redefinită în **CActivityList** încarcă sau salvează un obiect de la/în arhivă.

■ **CActivityList** - include următoarele metode procedurale definite în Sistemul Expert **PManager**:

- *DeleteElement* - este o metodă procedurală prin care se şterge activitatea (nodul) din listă care se află pe poziția primită ca parametru.
- *CleanUp* - este o metodă procedurală care şterge toate activitățile
- *ChekConsistency* - este metodă procedurală prin care se verifică dacă datele au fost introduse corect.

- *CheckPolicy* - este o metodă procedurală care verifică dacă pentru toate activitățile (nodurile) s-au introdus toate valorile duratei : optimist, probabil, pesimist şi dacă pentru aceste valori s-a calculat timpul mediu şi dispersia.
- *GetOptimal* - este o metodă procedurală care returnează valoarea optimă a

  unei activități.
- *CPMMethod* - apelează metodele de calcul a drumului critic şi afişează acest drum.
- *GetDen* - este o metodă procedurală prin care se returnează denumirea activității având sursa şi destinația transmise ca parametru.
- *Calculation* - este o metodă procedurală privată prin care se inițializează matricea work în funcție de parametru primit şi calculează drumul critic, adică drumul cu durata cea mai mare.
- *Calc* - este o metodă procedurală privată prin care se determină efectiv

  drumul critic.
- *Path* - este o funcție privată în care se pune drumul critic în lista path.
- *Result* - realizează afişarea drumului critic.

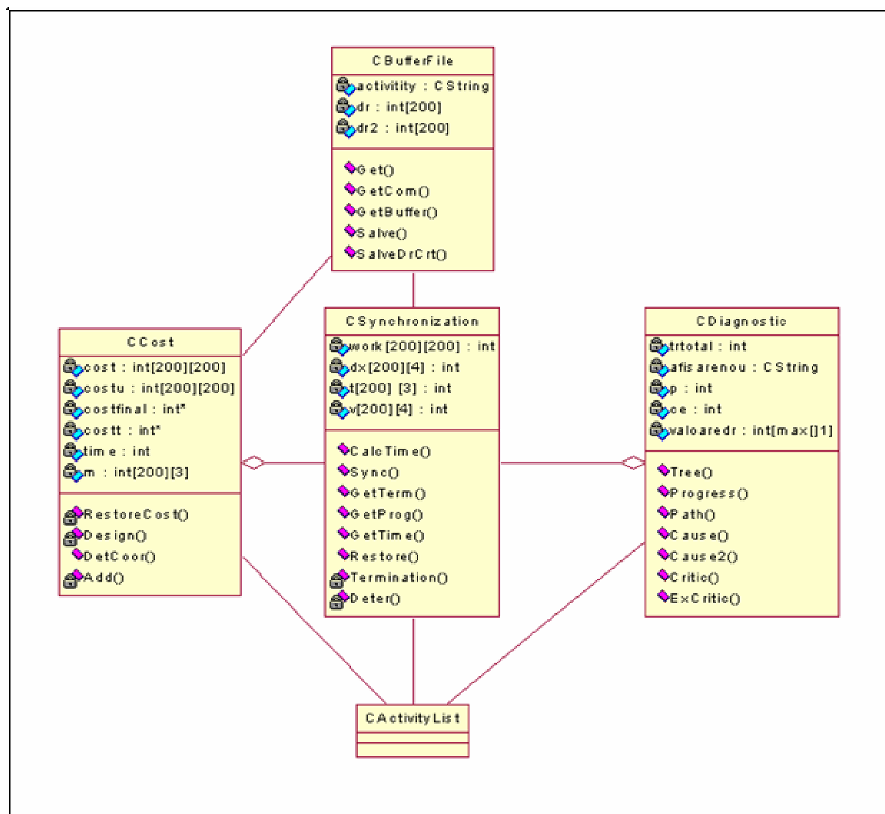


Fig. 3.4. PManager - Detaliu (b) din diagrama UML

- **CSynchronization** - reprezintă o clasă care realizează sincronizarea, (Fig.3.4). Verifică dacă o activitatea a fost terminată, dacă este în progres, sau dacă nu a început, vizualizând fiecare situație prin intermediul uni simbol. **CSynchronization** include următoarele metode definite în cadrul Sistemului Expert **PManager**:

- *CalcTime* - calculează durata care s-a scurs de la începerea proiectului.
- *Termination* - este o metodă privată, care verifică dacă toate activitățile care ajung în nodul primit ca parametru s-au terminat.
- *Restore* - reinițializează matricea **work** în funcție de selecția primită ca parametru cu valorile optim, probabil, pesimist în funcție de parametru. Dacă parametrul cu care este apelat constructorul este „0” atunci în matrice se regăseşte valoarea duratei optime, pentru „1” valoarea duratei pesimiste, iar pentru „2” valoarea duratei probabile.
- *Determ* - este o metodă privată care identifică care activitate este terminată, care este în progres şi care nu.
- *GetTerm* - returnează un pointer către matricea care memorează toate activitățile terminate.
- *GetProg* - returnează un pointer către matricea care a stocat toate activitățile care sunt în progres.
- *Sync* - este o metodă publică prin intermediul căreia se realizează sincronizarea: se inițializează vectorul v, se apelează metoda CalcTime, se apelează metoda Determ, afişându-se pentru toate activitățile semnul corespunzător stării în care se află: „!” - dacă este terminată, „<” - dacă este în curs de desfăşurare şi „ ” - dacă este nestartată.

■ **CCost** - reprezintă o clasă care realizează calcularea efectivă a coordonatelor de desenare pentru costuri. **CCost** furnizează următoarele metode definite în cadrul Sistemului Expert **PManager**:

- *RestoreCost* - este o metodă privată care inițializează matricea de cost, calculând costul pe unitate de timp.
- *Design* - este o metodă privată care calculează matricea costt.
- *DetCoor* - calculează efectiv coordonatele.
- *Add* - este o metodă care calculează cat s-a cheltuit pe o oră/zile/luni.

■ **CBufferFile** - reprezintă o clasă care lucrează cu fişierele “.dat” şi “.crt”. **CbufferFile** include următoarele metode definite în Sistemul **PManager**:

- *Get* - este o metodă care explorează în fişierul cu extensia „.dat” dacă la activitatea s-a făcut transfer de buffer.
- *GetBuffer* - returnează valoarea transferului de timp, dacă pentru o activitate din fişierul cu extensia „.dat” s-a făcut transfer de buffer.
- *GetCom* - explorează în fişierul cu extensia „.dat” dacă la activitatea cu denumirea primită ca parametru s-a făcut transfer de buffer şi returnează motivele pentru care s-a făcut acest transfer.
- *Salve* - este o metodă prin care se scrie în fişierul cu extensia „.dat” o nouă activitate la care s-a făcut transfer de buffer.
- *SalveDrCrt* - este o metodă prin care se scrie în fişierul cu extensia “.crt” drumul critic după ce la o activitate s-a făcut transfer de buffer.

■ **CDiagnostic** - reprezintă o clasă în care se realizează diagnosticul asupra unei activități. **CDiagnostic** include următoarele metode definite în Sistemul Expert **PManager**:

- *Tree* - este o metodă procedurală prin care se determină prin ce stări a trecut o activitate.
- *Progress* - este o metodă procedurală prin care se determină dacă o activitate este în progres, cat la sută s-a realizat din ea, este finalizată sau nefinalizată.
- *Critic* - verifică dacă activitatea primită ca parametru este critică sau nu.
- *ExCritic* - este o metodă în care se verifică dacă o activitate a fost critică.
- *Cause* - este o metodă care motivează de ce o activitate este critică. Dacă returnează „1” înseamnă că s-a făcut tranfer de buffer activității în cauză, dacă returnează „2” înseamnă că s-a făcut transfer de buffer unei alte activități din lanțul critic.
- *Cause2* - este o metodă prin care se verifică dacă o activitate a fost întârziată datorită transferului propriu de buffer, sau datorită transferului procesat în prealabil pentru una sau mai multe activități predecesoare.
- *Path* - este o procedură care determină toate lanțurile care pleacă din nodul inițial al grafului şi ajung în nodul primit ca parametru.

■ **CDate** - reprezintă o clasă în care se stochează data începerii proiectului (anul, luna, minut, ora, ziua), timpul alocat proiectului (pani, pminute, pore, pzile), programul de lucru (minsp, minst, orasp, orast) şi unitatea de timp a proiectului. **CDate** - include următoarele metode procedurale definite în Sistemul Expert **PManager**:

- *Reset* - este o metodă care resetează datele, adică la începerea unui nou proiect datele devin zero.
- *Serialize* - este o metodă prin care se salvează/încarcă în fişier extensia “.pmd”, datele care sunt conținute în această clasă.
- *GetUtp* - este o procedură care returnează unitatea de timp în care este setat proiectul.

■ **CAsisstant** (Fig. 8.5) - reprezintă o clasă în care se activează “*asistentul*” cu ajutorul căruia utilizatorul poate fi în dialog permanent cu Sistemul. Ea conține CAsisstant1 şi CAsisstant2 prin care se realizează interfata cu utilizatorul. CAsisstant include următoarele metode procedurale definite în Sistemul Expert **PManager:**

- *Status* - este o metodă care realizeză un diagnostic al proiectului. Atât pentru activitățile finalizate cât şi cele în curs de desfăşurare se creează un obiect al clasei Ciagnostic apelând metoda Tree. (§ 7.2.5)
- *ProgBuffer* - este o metodă care calculează transferul de buffer necesar pentru o activitate care este în întârziere.
- *Asisstent - este o metodă care realizează dialogul cu utilizatorul.*

■ **CleftView** (fig. 3.6) - reprezintă o clasă care moşteneşte clasa **CTreeView**, care la rândul ei reprezintă o clasă simplificată care utilizează *arborele de control.*
CleftView includ următoarele metode procedurale redefinite în Sistemul Expert **PManager**:

- *OnClickAddActivity* - este o metodă protejată, care este apelată când se apasă butonul „adăugare activitate”, adăugându-se astfel o activitate în listă.

- *OnClickDelActivity* - este o metodă protejată care este apelată atunci când se apasă butonul „ştergere activitate”, situaţie în care se şterge activitatea introdusă ca parametru.
- *OnClickProgramLc* - este o metodă protejată prin care se apelează “programul de lucru” şi care returnează valorile setate într-o instanţă a clasei **CData**.
- *OnClickTPlanT* - este o metodă protejată care se apelează atunci când se apasă butonul “data începerii proiectului”.
- *OnClickTTime* - este o metodă care se apelează când se apasă butonul“Timpul planificat” de introducere a estimării proiectului.
- *OnClickTBuffer* - este o metodă protejată care se apelează când se doreşte transfer de buffer.
- *OnClickGrafic* - este o metodă protejată care se apelează când se doreşte să se afişeze graficul costurilor.

■ **CMainFrame -** reprezintă clasa care moşteneşte **CFrameWnd**, care este o clasă care implementează funcţionalitatea interfeţei pentru Windows (SDI), cu meniuri şi pop-up în ferestre. CMainFrame are următoarele metode moştenite:

- *AssertValid* - este o metodă care validează integritatea obiectelor.
- *Dump* - este o metodă care dizlocă din memorie obiectul.

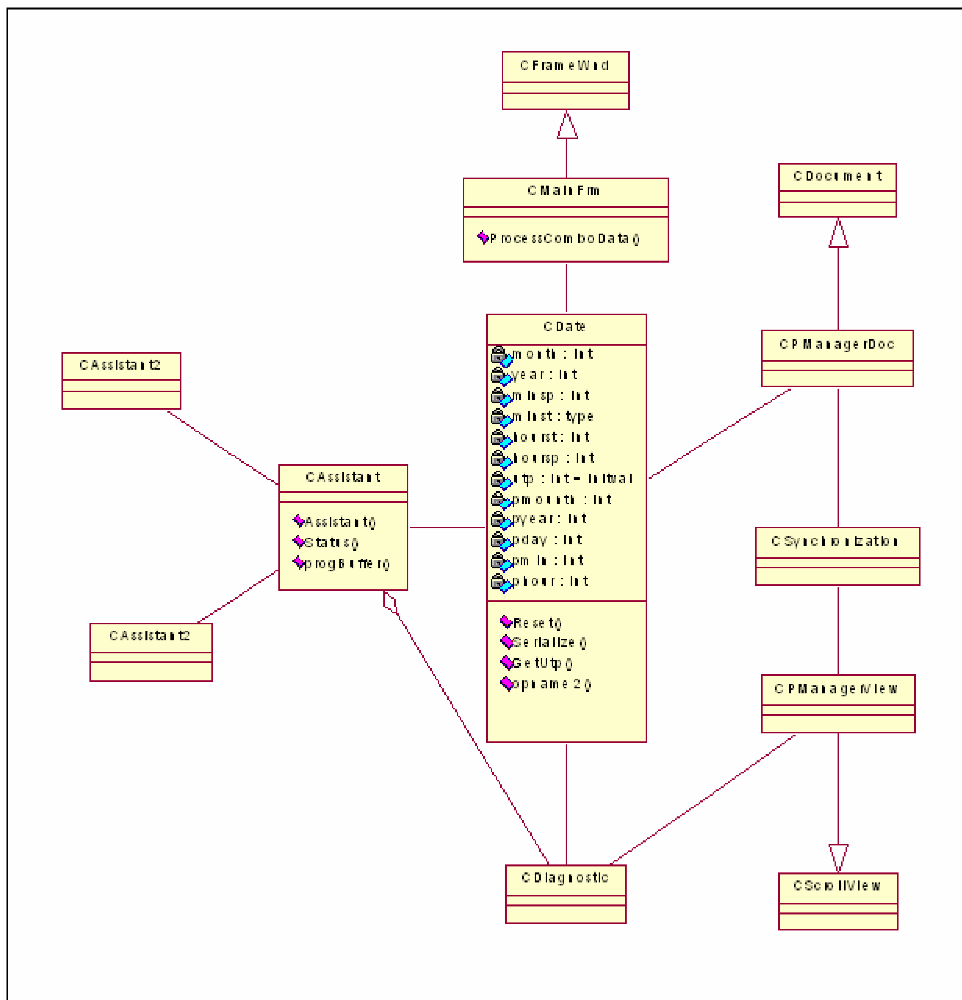


Fig. 3.5. PManager - Detaliu (c) din diagrama UML

- **MainFrame** include metodă procedurală definită în **PManager** :
  - *ProcessComboData* - este o metodă care calculează *probabilitatea de finalizare* şi *afişează valorile drumului critic.*
- **CPManagerApp** – reprezintă o clasă care moşteneşte clasa CWinApp.
- **CWinApp** - reprezintă o clasă de bază care poate fi moştenită pentru a deriva obiecte de aplicaţie pentru Windows.

  CWinApp include următoarele metode procedurale:
  - *InitInstance* - este o metoda care permite rularea mai multor instanţe ale aceluiaşi program simultan. Returnează o valoare diferită de 0 dacă a fost iniţializată
  - *OnAppAbout* - este o metodă care afişează informaţii despre program.
  - *OnAppAsistent* - este o metodă prin care se apelează cererea de „*asistent”*.
- **CPManagerDoc** - reprezintă o clasă care este derivată din **Cdocument**, care la rândul ei implementează funcţionalităţi de bază pentru clase care utilizează documente. Un document reprezintă un “unit” de date şi este implementat în această clasă cu deschidere

de fişier, având comanda File, Open şi salvarea, având comanda File Save. **CPManagerDoc** include următoarele metode moştenite de la clasa **CDocument**:

- *AssertValid* - este o metodă în care se validează integritatea obiectelor.
- *Dump* - este o metodă care şterge din memorie obiectul.

■ **CManagerDoc** include următoarele metode definite în Sistemul **PManager**:

- *OnNewDocument* - *e*ste o metodă prin care se şterg toate activitățile şi se resetează obiectul din clasa **CDate**. Prin această metodă se creează un nou proiect.
- *OnOpenDocument* - este o metodă prin care se deschide un proiect.
- *OnSaveDocument* - este o metodă prin care se salvează in arhivă documentul.
- *Serialize* - este o metodă definită în clasa CObject şi redefinită, care încarcă sau salvează un obiect de la/în arhivă.

■ **CDialog** - reprezintă clasa de bază pentru afişarea căsuțelor de dialog pe ecran.

Clasa **CDialog** este moştenită de următoarele clasele: **CAssistant1**, **CAssistant2**, **CAssistant3**, **CAssistant4**, **CAssistant5**, **CAssistant6**, **CAssistant7**, **CBazaDlg**, **CDeleteDlg**, **CDiagnosticDlg**, **CGrafigDlg**, **CPTimeDlg**, **CBufferDlg**, **CTimeDlg**.

Toate aceste clase sunt utilizate pentru a realiza dialogul cu utilizatorul. Pe lângă metodele moştenite de la clasa **CDialog** mai există o procedură prin care se realizează fie citirea din căsuța de dialog fie se realizează scrierea în componentele căsuței de dialog.

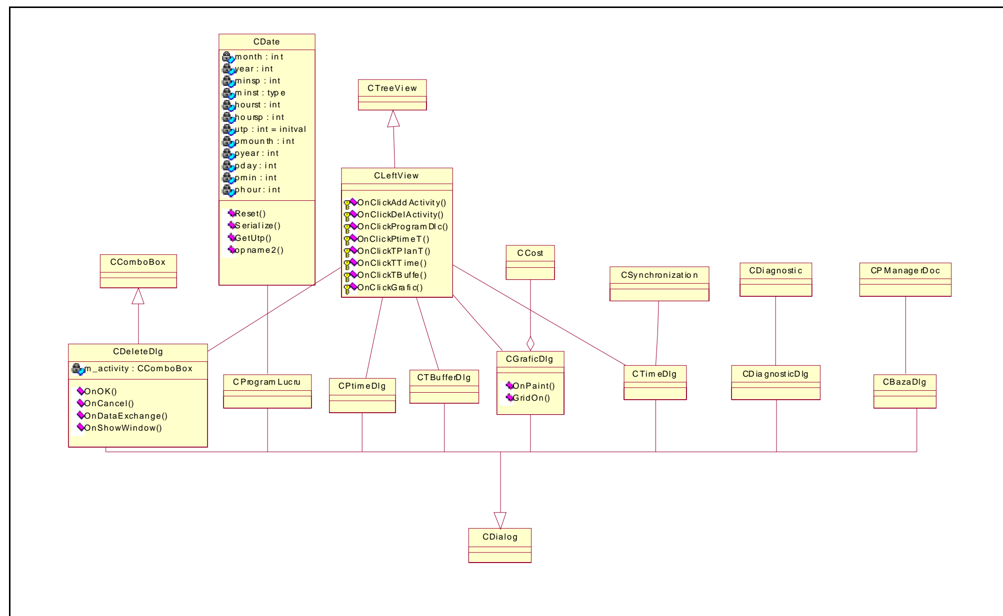


Fig.3.6. PManager - Detaliu (d) din diagrama UML

3.3.1. Reutilizarea codului

În proiectele produselor program este indicat să se reutilizeze codurile existente, fiind astfel necesară o proiectare cât mai generală a claselor pentru a putea fi refolosite.

În cazul Sistemului Expert "**PManager**" reutilizarea de cod s-a dovedit utilă în special la calcularea drumului critic, astfel încât, prin intermediul unui anumit parametru, această procedură de calcul este în mod repetat apelată. Un exemplu de reutilizare este calcularea coordonatelor de trasare a graficului evoluției costurilor într-o clasă CCost. De asemenea, clasa care realizează sincronizarea impune o reutilizare de cod, având în vedere că sincronizarea este necesară la fiecare schimbare a datelor.

Principalul avantaj al reutilizării de cod este acela că eventualele modificări se realizează într-un singur loc.

3.4. Implementarea codului

La baza construirii modelului bazei de cunoştinţe, au stat piesele de cunoaştere furnizate de următorii 5 algoritmi numiţi şi "proceduri de calcul aplicate cunoştinţelor", reprezentând ansamblul formalismului de reprezentare asociat strategiei de inferenţă a Sistemului Expert **PManager**:

1. *Calculul Drumului Critic,*
2. *Sincronizarea* proiectului,
3. Algoritmul realizat pentru *Transferul de Buffer*,
4. Algoritmul *analizei diagnostic* a evoluției activităților proiectului.
5. Algoritmul pentru *trasarea graficului* evoluției costurilor .

În continuare se prezintă aspectele esențiale legate de implementarea acestor modele.

1

Calculul Drumului Critic

a

Prezentarea algoritmului

Drumului Critic, reprezintă secvența activităților cu durată maximă de timp cuprinsă între evenimentul inițial şi evenimentul final al proiectului.

Acest algoritm este inspirat din algoritmul lui Floyd [Cre-92] pentru calculul drumului minim într-un graf, fiind reconceput pentru *calculul drumului maxim într-un graf*.

Algoritmul utilizează o matrice de adiacență "**work**". Având în vedere faptul că duratele activităților sunt mai mari sau egale cu "0" s-a stabilit prin convenție ca:

- în starea inițială (înainte de a fi activate datele specifice unui proiect) toate elementele matricii "Work" să fie egale cu "-1";
- dacă există drum între nodul **i** şi nodul **j** atunci **work**[i][j] este egal cu valoarea optimistă/probabilă/pesimistă/medie a activității respective.

Algoritmul poate fi definit astfel: *se execută max. iterații asupra matricii "work", iar după iterația k, fiecare locație din matricea "work" va conține lungimea maximă a oricărui drum la nodul* ***i*** *la nodul* ***j****, drum care să nu parcurgă vre-un nod cu indicele mai mare decât* ***k****.*

În fig 3.7 este prezentată şi analizată organigrama algoritmului de calculul al drumului critic în matricea "work", considerat ca fiind adecvat implementării Sistemului Expert "**PManager**".

Nodurile **i** şi **j** pot fi oricare două noduri ale grafului care delimitează începutul şi sfârşitul drumului, iar **k** oricare nod intermediar mai mic. Pentru calculul matricii work se utilizează relația 3.1: (Fig. 3.8)

$$Work_k[i][j]=maxim(work_{k-1}[i][j],\ work_{k-1}[i][k]+work_{k-1}[k][j]),\ (Fig.3.8) \tag{3.1}$$

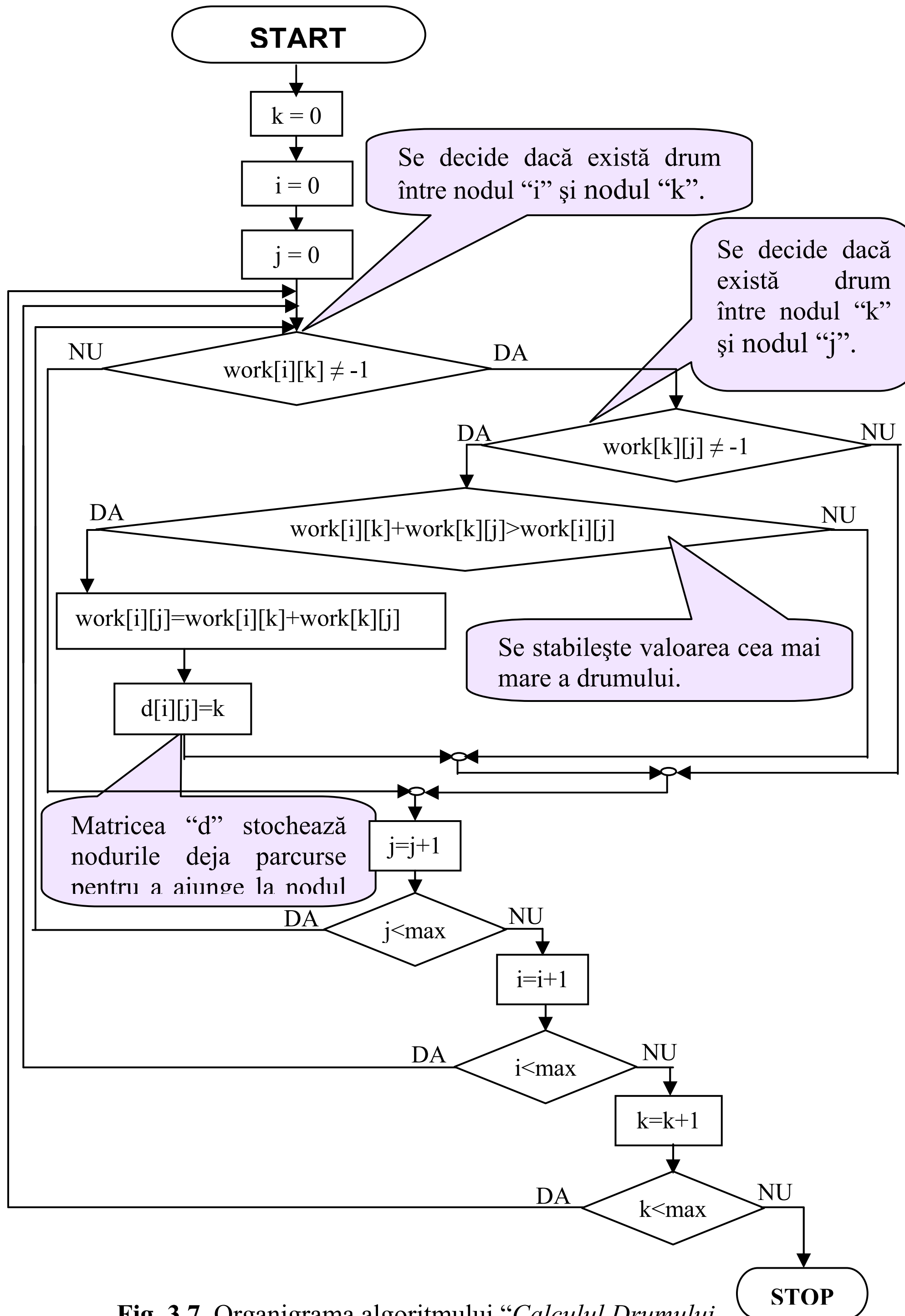


Fig. 3.7. Organigrama algoritmului "*Calculul Drumului Critic*"

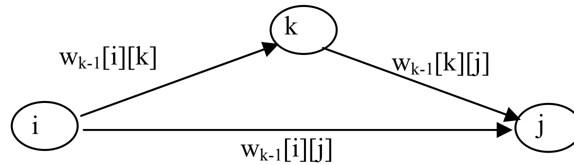


Fig. 3.8. Detaliu din cadrul rețelei unui proiect

unde k reprezintă momentul la care se calculează matricea **work**. Matricea **work** este unică.

Interpretarea relației (3.1) pentru calcul drumului de la nodul **i** la nodul **j** (**work$_k$**[i][j]) se realizează astfel: Se compară **work$_k$**[i][j] – valoarea drumului de la **i** la **j** fără să treacă prin nodul **k**, cu suma **work$_{k-1}$**[i][k]+ **work$_{k-1}$**[k][j] – valoarea drumului de la **i** la **k** plus valoarea drumului de la **k** la **j**.

Dacă drumul este mai lung se atribuie lui **work$_k$**[i][j] valoarea **work$_{k-1}$**[i][k] + **work$_{k-1}$**[k][j], în caz contrar valoarea lui **work$_k$**[i][j]=**work$_{k-1}$**[i][j] rămâne neschimbată.

Algoritmul în pseudocod este următorul:

```
for(k=0;k<max;k++)
        for(i=0;i<max;i++)
            for(j=0;j<max;j++)
                if ((work[i][k]!=-1)&&(work[k][j]!=-1)){
                    if ((work[i][k]+work[k][j]>work[i][j])){
                        work[i][j] = work[i][k] + work[k][j];
                        d[i][j]=k; //drumul
                    }
                }
```

Valoarea drumului critic de la nodul “0” la nodul “max” este stocată în matricea **work** la **work**[0][max].

Drumul (nodurile deja parcurse pentru a ajunge la nodul “max”) se stochează într-o matrice “**d**”. Matricea **d**[i][j] se complectează cu valoarea nodului prin care trece drumul cel mai lung de la **i** la **j**.

b Strategia de control în spațiul stărilor pentru calculul Drumului Critic

Considerând rețeaua activităților unui proiect de forma reprezentată în fig 3.9.

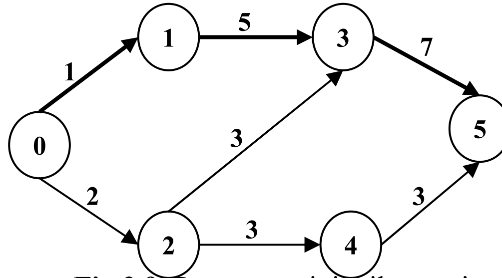


Fig 3.9. Rețeaua activităților unui proiect

În continuare se prezintă strategia de căutare a Drumului Critic în matricea **work**. Pe măsura introducerii datelor în sistem, matricea **work** devine:

k, j →

i, k ↓

$$\text{work} = \begin{pmatrix} -1 & 1 & 2 & -1 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix}$$

Se verifică condiția de stabilire a drumului cu valoarea cea mai mare:

work[i][k] + work[k][,j] > work[i][j]

Pentru:

i = 0,
k = 0,
j = 0

⇒ work[0][0] + work[0][0]>work[0][0]

Se observă că nu au fost identificate drumuri între noduri.

⇒ i = 0,
k = 0,
j = 1,…5

Se incrementează "j " și se observă aceeași situatie până la j = 5, după care se incrementează "k".

⇒ i = 0,
k = 1,
j = 0,…3

Se incrementează "j" rezultând aceeași situație până la j = 3 pentru care se identifică drumuri între noduri (succesiune de activitati): 0 → 1 → 3. Aplicând condiția (3.1) rezultă:

work[i][k] + work[k][j] > work[i][j] ⇒

work[0][1] + work[1][3] > work[0][3] ⇒

$1 + 5 \;\; > \; -1 \;\; \Rightarrow 6 > -1$

Elementul **work**[0][3] = -1 al matricii **work**[i][j] va fi înlocuit cu **work**[0][3] = 6 rezultând următoarea matrice $\mathbf{work_1}$[i,j].

$$work = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{pmatrix} -1 & 1 & 2 & -1 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix} \end{array} \qquad work_1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{pmatrix} -1 & 1 & 2 & 6 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix} \end{array}$$

$\Rightarrow$ i = 0, k = 1, j = 1,…5

$\Rightarrow$ i = 0, k = 2, j = 0,…3

Se incrementează "j " până la j = 5, situații pentru care nu s-au găsit drumuri între noduri, după care se incrementează k=2 şi j= 0,…3, situație pentru care se identifică drumurile (succesiune de activitati): $0 \rightarrow 2 \rightarrow 3$. Se verifică condiția (3.1) pentru a stabili eventuala trecere în următoarea stare.

$work[i][k] + work[k][j] > work[i][j] \Rightarrow$

$work_1[0][2] + work_1[2][3] > w_1[0][3] \Rightarrow$

$2 + 3 \;\; > \;\; 6$

Elementul $work_1[0][3]$ =6 al matricii $work_1[i][j]$ este mai mare decât suma $work_1[0][2] + work_1[2][3] = 5$, ca urmare se selectează $work_1[0][3]$ =6 şi se avansează la

$\Rightarrow$ i = 0, k = 2, j = 4

Se incrementează j = 4 pentru care se identifică drumurile (succesiune de activități): $0 \rightarrow 2 \rightarrow 4$. Aplicând condiția (3.1) rezultă următoarea situatie.

următoarea stare:

$work_1[0][2] + work_1[2][3] = 5$, matricea $work_1$[i,j] rămânând neschimbată.

$work[i][k] + work[k][j] > work[i][j] \Rightarrow$

$work_1[0][2] + work_1[2][4] > work_1[0][4] \Rightarrow$

$2 + 3 \;\; > \;\; -1 \;\; \Rightarrow 5 > -1$

$$
work_2 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{pmatrix} -1 & 1 & 2 & 6 & 5 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix} \end{array}
$$

$i = 0$,
$\Rightarrow k = 2$,
$j = 5$

$i = 0$,
$\Rightarrow k = 3$,
$j = 0,\dots 1$

Se incrementează j = 5, situație pentru care nu s-au identificat drumuri între noduri, după care se incrementează k=3 şi j= 0,…5, ultima situație fiind cea pentru care se identifica drumurile (succesiune de activitați): $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$. Se verifică condiția (1) pentru a stabili eventuala trecere în următoarea stare.

work[i][k] + work[k][j] > w[i,j] $\Rightarrow$

$work_2[0,3] + work_2[3,5] > work_2[0,5] \Rightarrow$

$6 + 7 > -1 \Rightarrow 13 > -1$

Valoarea maximă atribuită drumului critic este 13, fiind regăsită în matricea $work_3[0][5]$.

$$
work_3 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{pmatrix} -1 & 1 & 2 & 6 & 5 & 13 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix} \end{array}
$$

2 *Sincronizarea proiectului*

Sincronizarea evoluției duratelor de timp ale activităților din rețeaua proiectului se realizează în concordanță cu ceasul sistemului având la dispoziție următoarele informații furnizate de către utilizator:

- data începerii proiectului
- programul de lucru
- ora la care va fi startat proiectul
- unitatea de timp aleasă pentru planificarea activităților (minute, ore, zile, luni)

Sincronizarea proiectului se realizează în clasa **CSynchronization**.

a

Prezentarea algoritmului

Algoritmul utilizat în procesul de sincronizare se bazează pe strategia de căutare de *back-tracking recursiv*. Acest algoritm identifică într-o matrice **work** activitățile *finalizate* şi *în curs de desfăşurare*.

Pentru o sincronizare corectă se verifică următoarea condiție: pentru validarea startului unei activități din nodul "i" să se confirme finalizarea tuturor predecesoarelor în acel nod. Aceasta se realizează prin intermediul metodei "*Termination"*.

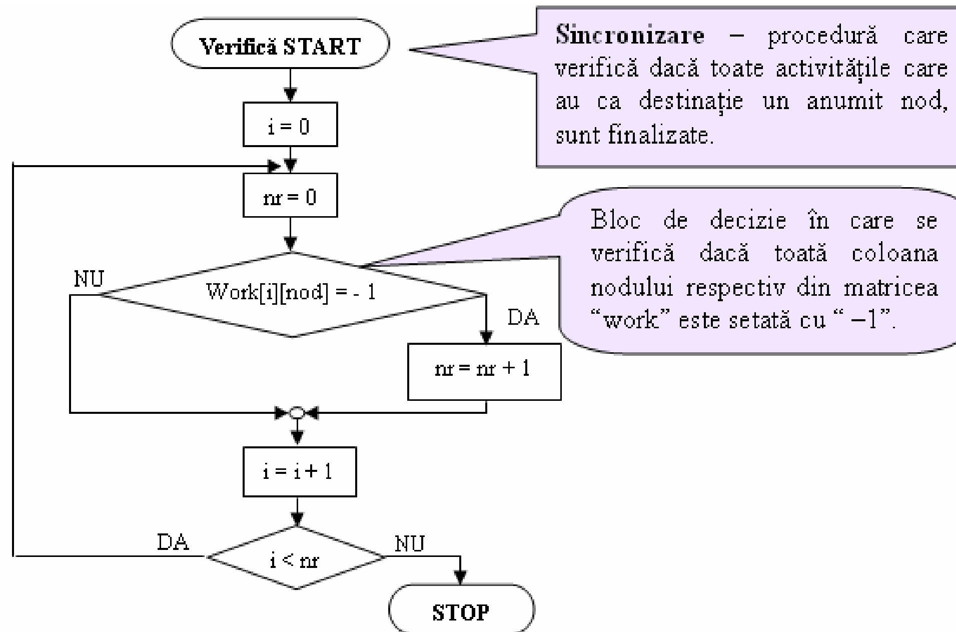


Fig.3.10. Organigrama algoritmului "*Termination*"

În fig. 3.10 este prezentată şi comentată organigrama algoritmului de verificare *"Termination"*, considerat ca fiind adecvat implementării Sistemului Expert "**PManager**".

Atunci când o activitate i → j este finalizată, matricea **work**[i][j] este setată pe valoarea "–1", marcând astfel terminarea activități i → j.

Metoda "*Termination"* stabileşte dacă toate activitățile care ajung în nodul **i** sunt terminate. Această metodă este concepută astfel: *dacă toată coloana nodului respectiv este setată cu " –1" înseamnă că toate activitățile care ajung în nodul **i** sunt terminate, în caz contrar există activități care nu sunt încă finalizate.*

Metoda de sincronizare parcurge în continuare paşii unui nou algoritm de determinare a activităților *finalizate* sau *în curs de desfăşurare*. Paşii şi organigrama acestui algoritm sunt prezentați în lista următoare şi în fig. 3.11.

Paşii algoritmului de determinare:

1. se balează matricea pe linie (incrementând indicele de coloană), pentru nodul primit ca parametru.

2. dacă există drum de la nodul primit ca parametru **nod** la nodul **i** (work[nod][i]>=0) atunci se calculează o valoare intermediară a drumului de la nodul inițial "0" la nodul **i**. Valoarea

intermediară se calculează adăugând valoarea drumului de la sursa inițială “0” la nodul **nod.** La această valoare se adaugă valoarea drumului de la **nod** la **i**.

3. se compară dacă valoarea rezultată este mai mare decât valoarea drumului de la nodul inițial “0” la nodul **i**, care nu trece prin nodul **nod**. În caz afirmativ se selectează valoarea maximă rezultată de la nodul inițial “0” la nodul **i**, trecând însă prin nodul **nod**.

- Se utilizează în plus un vector **v[i],** în care se stochează valoarea drumului maxim de la nodul inițial “0” la nodul **i**.
- “**Cat**” reprezintă o variabilă care preia valoarea timpului scurs de la începerea proiectului.
- Matricea **dx**[k1][4] este o matrice care este tranzitată de activitățile în curs de desfăşurare. Astfel pe poziția dx[i][0] se identifică nodul sursă al activității, pe poziția dx[i][1] se identifică nodul destinației, pe poziția dx[i][2] se identifică cât s-a realizat din activitate, pe poziția dx[i][3] se identifică când începe activitatea respectivă.
- Matricea **t[k2]** este matricea în care sunt memorate activitățile terminate.

Algoritmul în pseudocod este:

```
Determ(nod, int valdrum)
        pentru i=0 la numărul de activitati
                dacă work[nod][i]>=0 (există drum de la nodul nod la nodul i) atunci
                        val1=v[nod]+work[nod][i];
                        daca (val1>v[i]) atunci v[i]=val1;
                                daca(val1>cat) atunci
                                        daca verifica(nod)=1 (verifică dacă toate activitățile care
ajung în nodul nod s-au finalizat) atunci
                                                dx[k1][0]=nod;
                                                dx[k1][1]=i;
                                                dx[k1][2]=cat-v[nod];
                                                dx[k1][3]=v[nod];
                                                k1++;
                        altfel
                                        t[k2][2]=v[nod];
                                        work[nod][i]=-1;
                                        t[k2][0]=nod;
                                        t[k2][1]=i;
                                        k2++;
dacă verifica(i)=1(dacă toate activitățile care ajung în nodul i s-au finalizat) atunci
```

determina(i,v[i]);

verifica(int nod)

pentru i=0 de la numărul de noduri

dacă (work[i][nod]=-1) atunci

nr++;

dacă (nr=numărul de noduri)   val= 1;

altfel 0;

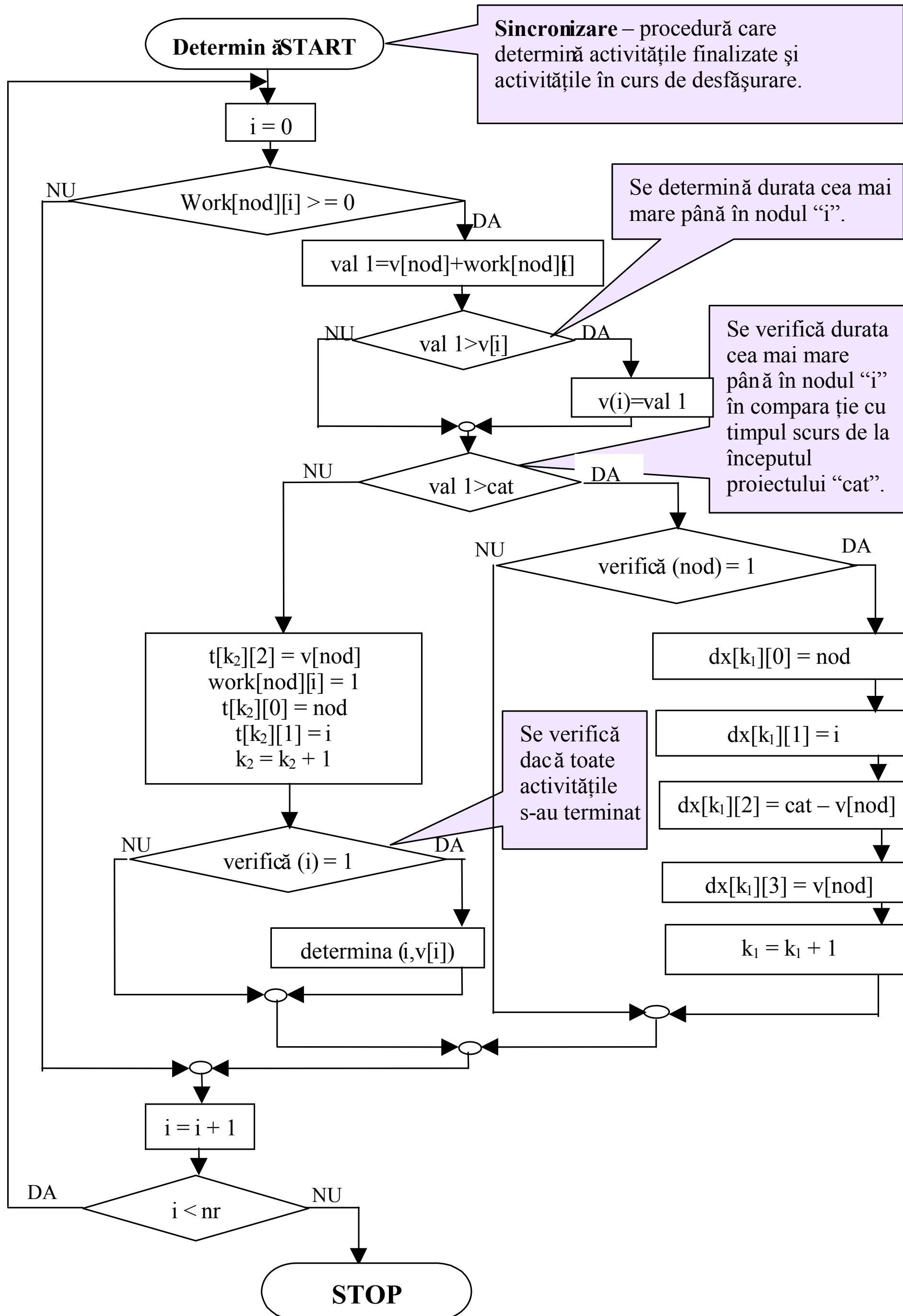


Fig. 3.11 Organigrama algoritmului "*determină*"

b

Strategia de control în spațiul stărilor pentru procedura de sincronizare

Considerând rețeaua din fig. 3.9 în continuare se abordează problema sincronizării evoluției acestui proiect.

Sincronizarea se realizează în acord cu secvența duratelor optimiste. Matricea **work** se actualizează în felul următor:

- *indicelui (*i*) de linii ale matricii **work** îi corespund nodurile sursă ale activităților din rețea, iar indicelui (j) de coloane îi corespund nodurile destinație ale activităților din rețea, activitățile corespunzătoare rețelei fiind preluate dintr-o listă* AList*.*
- *valoarea optimistă a duratelor activităților este stocată în matricea work[i][j]*

  *daca nu există drum intre doua noduri (**i** și **j**) atunci matricea se completează cu valoarea "–1"*

Pentru rețeaua din fig. 3.9, matricea work devine:

$$\textbf{work} = \begin{pmatrix} -1 & 1 & 2 & -1 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix}$$

În continuare trebuie calculat timpul scurs de la începutul proiectului. Acest lucru se realizează cu ajutorul metodei procedurale "*CalTime*".

Pentru exemplificare se presupune că proiectul

- a inceput pe data de 27 mai 2002 la ora 17
- ne aflam în 28 mai 2002, ora 16 (ceasul sistem)
- programul de lucru este de la ora 8 la18.
- metoda *CalculareTimp* calulează timpul memorându-l într-o variabilă **cat**.

În cazul acestui exemplu, variabila **cat** are valoarea 1+8=9 ore (1 oră efectuată în 27 mai de la ora 17 la 18 şi 8 ore efectuate în 28 mai de la ora 8 la 16).
După ce s-a calculat timpul scurs din 27 mai 2002, ora 17 până în 28 mai ora 16 se apelează metoda procedurală *determină(nod, valoare)* care stabileşte dacă:

- activitatea a fost finalizată,
- activitatea este în progres
- activitatea este neîncepută.

Metoda se apelează recursiv.

Inițial nod=0 și valoare=0 si un vector de 5 elemente **v** care este initializat cu "0", matricea **t** este "0" si vectorul **dx** este "0", k1=0, k 2=0.

Nod=0; i=0;

Work[nod][i]=-1<0 deci nu se întâmplă nimic.

Nod=0; i=1

Daca work[nod][i]=1>0 (există drum de la nod la j) atunci

val1=v[nod]+work[nod][i]=0+1=1

daca val1>v[i](1>0) atunci v[i]=val1

vectorul **v** este de forma:

$$\mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

daca val1<cat=9 atunci activitatea 0→1este finalizată si acest lucru se memorează într-un vectror t[k2][0]=nod, t[k2][1]=i, t[k2][2]=v[nod]=0, k2=k2+1

Matricea **t** va fi de forma:

$$\mathbf{t} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

iar valoarea work[nod][i]=-1.

Matricea **work** are forma:

$$\mathbf{work} = \begin{pmatrix} -1 & -1 & 2 & -1 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix}$$

Se verifică dacă toate activitățile care ajung în nodul **i** s-au finalizat (acest lucru se realizează cu ajutorul metodei *verifica* care stabileşte daca matricea are în toată coloana **i** valoarea "–1"; daca da înseamnă că toate activitățile care au destinația **i** s-au finalizat).

Daca toate activitățile care ajung la destinația **i** s-au finalizat atunci se apelează metoda *determina*(i,v[i]), adica *determina*(1,1)

În acest caz nod =1 si i=0 la 5

Pentru nod=1, i=0..2 nu se întâmpla nimic

Nod=1, i=3 work[1][3]=5>0 atunci val1=v[1]+work[1][3]=1+5=6

daca val1>v[3] v[3]=val1

Vectorul v devine: $\mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 6 \\ 0 \\ 0 \end{bmatrix}$

val1<cat(6<9), activitatea 1→3 este finalizată si t[k2=1][0]=nod=1, t[k2=1][1]=i=3, t[k2=1][2]=v[nod]=v[1]=1, k2=k2+1

Matricea t va avea forma: $\mathbf{t} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Matricea Work [nod][i]=work[1][3]=-1, devine>

$$\mathbf{work} = \begin{pmatrix} -1 & -1 & 2 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix}$$

Se verifică dacă toate activitățile care au destinația 3 s-au finalizat. Se observă din matricea **work** activitatea 2 →3 nu s-a finalizat (este în derulare).

Nod=1; i=4,5 nu se întamplă nimic

Se revine la nod =0 si i=2.

work[0][2]=2>0 atunci val1=v[0]+work[0][2]=0+2=2

daca val1>v[2] v[2]=val1=2

Vectorul v devine:

$$\mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 6 \\ 0 \\ 0 \end{bmatrix}$$

val1<cat(2<9), activitatea $0 \rightarrow 2$ este finalizată si t[k2=2][0]=nod=0, t[k2=2][1]=i=2, t[k2=2][2]=v[nod]=v[0]=0, k2=k2+1

Matricea t devine la rândul ei:
$$\mathbf{t} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Matricea work [nod][i]=work[0][2]=-1, va fi de forma:

$$\mathbf{work} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 3 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix}$$

Se verifică dacă toate activitățile care au destinația “2” s-au finalizat. Se observă din matricea **work,** toate activitățile care ajung în destinația “2” s-au finalizat.

În continuare se metoda procedurală *determina*(2,2).

În situația nod=2, i=0..2, nu se întâmplă nimic, caz în care se consideră că sau nu există secvență de activități între aceste noduri, sau că acestea s-au finalizat.

Pentru nod=2; i=3

Work[2][3]=3>0 atunci val1=v[2]+work[2][3]=2+3=5

dacă val1<v[3] (5<6), vectorul **v** rămâne neschimbat:
$$\mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 6 \\ 0 \\ 0 \end{bmatrix}$$

val1<cat(5<9) activitatea $2 \rightarrow 3$ este finalizată si t[k2=3][0]=nod=2, t[k2=3][1]=i=3, t[k2=3][2]=v[nod]=v[2]=2 k2=k2+1

Matricea t devine: $\mathbf{t}=\begin{bmatrix}0&1&0\\1&3&1\\0&2&0\\2&3&2\\0&0&0\\0&0&0\\0&0&0\end{bmatrix}$

Matricea work [nod][i]=work[2][3]=-1, la rândul ei devine:

$$\mathbf{work}=\begin{pmatrix}-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&3&-1\\-1&-1&-1&-1&-1&7\\-1&-1&-1&-1&-1&3\end{pmatrix}$$

Se verifică dacă toate activitățile care au destinația 3 s-au finalizat. Se observă din matricea work toate activitățile care ajung in destinația 3 s-au finalizat.

Apelează procedura determina(3,6)

În situația nod=3; i=0..4 nu se întâmpla nimic, considerându-se în acest caz că nu există secvență de activități între aceste noduri, sau că acestea s-au finalizat.

Nod=3; i=5

Work[3][5]=7>0 atunci val1=v[3]+work[3][4]=6+7=13

daca val1>v[5](13>0) v [5]=val1=13

Vectorul v devine în acest: $\mathbf{v}=\begin{bmatrix}0\\1\\2\\6\\0\\13\end{bmatrix}$

val1>cat(13>9)

Se verifică dacă toate activitățile care au destinația 3 s-au finalizat.

Activitatea 3→5 este în desfăşurare si dx[k1=0][0]=nod=3, dx[k1=0][1]=i=5, t[k1=0][2]=v[nod]=cat-v[3]=8-6=2 k1=k1+1

Matricea **dx** devine: $\mathbf{dx} = \begin{bmatrix} 3 & 5 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Nod=2 ; i=4

Work[2][4]=3>0 atunci val1=v[2]+work[2][4]=2+3=5

daca val 1>v[4](5>0) v[4]=val 1

Vectorul v va avea forma: $\mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 6 \\ 5 \\ 13 \end{bmatrix}$

val1<cat(5<9) activitatea 2→4 este finalizată şi t[k2=4][0]=nod=2, t[k2=4][1]=i=4, t[k2=4][2]=v[nod]=v[2]=2 k2=k2+1

Matricea t şi matricea Work [nod][i]=work[2][4]=-1, devin în acest caz:

$$\mathbf{t} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 2 & 0 \\ 2 & 3 & 2 \\ 2 & 4 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} ; \qquad \mathbf{work} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 \\ -1 & -1 & -1 & -1 & -1 & 3 \end{pmatrix}$$

Se verifică daca toate activităţile care au destinaţia 4 s-au finalizat. Se observă că răspunsul este afirmativ.

Aplează procedura *determina*(4,5)

În situaţia nod=4; i=0..4, nu se întâmpla nimic; în acest caz există secvenţă de activităţi între aceste noduri, sau acestea s-au terminat.

Pentru nod=4; i=5

Work[4][5]=3>0 atunci val1=v[3]+work[3][4]=8+3=11

val1<v[i]=v[5](11<13)

Vectorul v ramâne neschimbat: $\mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 6 \\ 5 \\ 13 \end{bmatrix}$

val1>cat(11>9) activitatea 4→5 este în desfăşurare si dx[k1=1][0]=nod=4, t[k1=1][1]=i=5, t[k1=1][2]=cat-v[nod]=cat-v[4]=9-5=4 k1=k1+1

Matrice dx devine în acest caz: $\mathbf{dx} = \begin{bmatrix} 3 & 5 & 2 \\ 4 & 5 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

În situația nod=2; i=5 nu se întâmpla nimic. În acest caz sau nu există secvență de activități între aceste noduri, sau că acestea s-au finalizat.

Se revine la nod=0; i=3..5 nu se întâmpla nimic, sau nu există secvență de activități între aceste noduri, sau acestea s-au finalizat.

3 *Transferul de Buffer*

a **Prezentarea algoritmului**

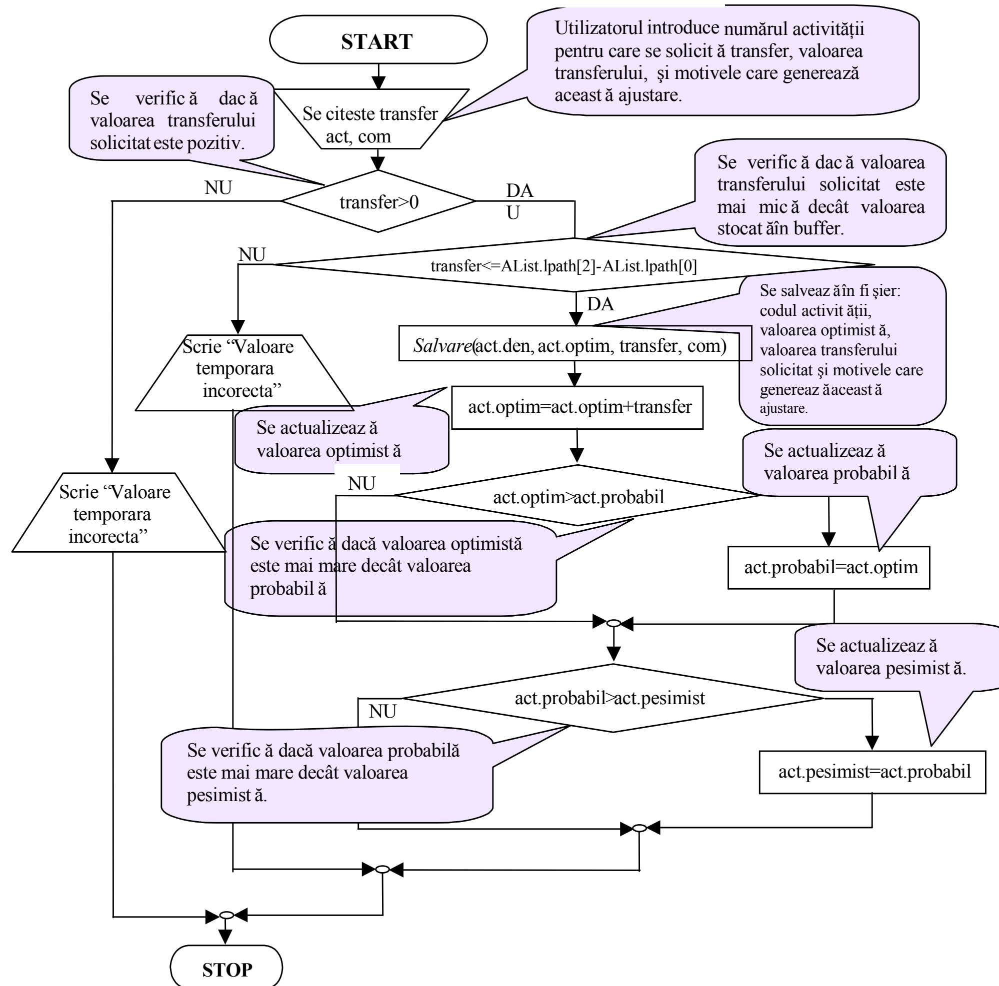
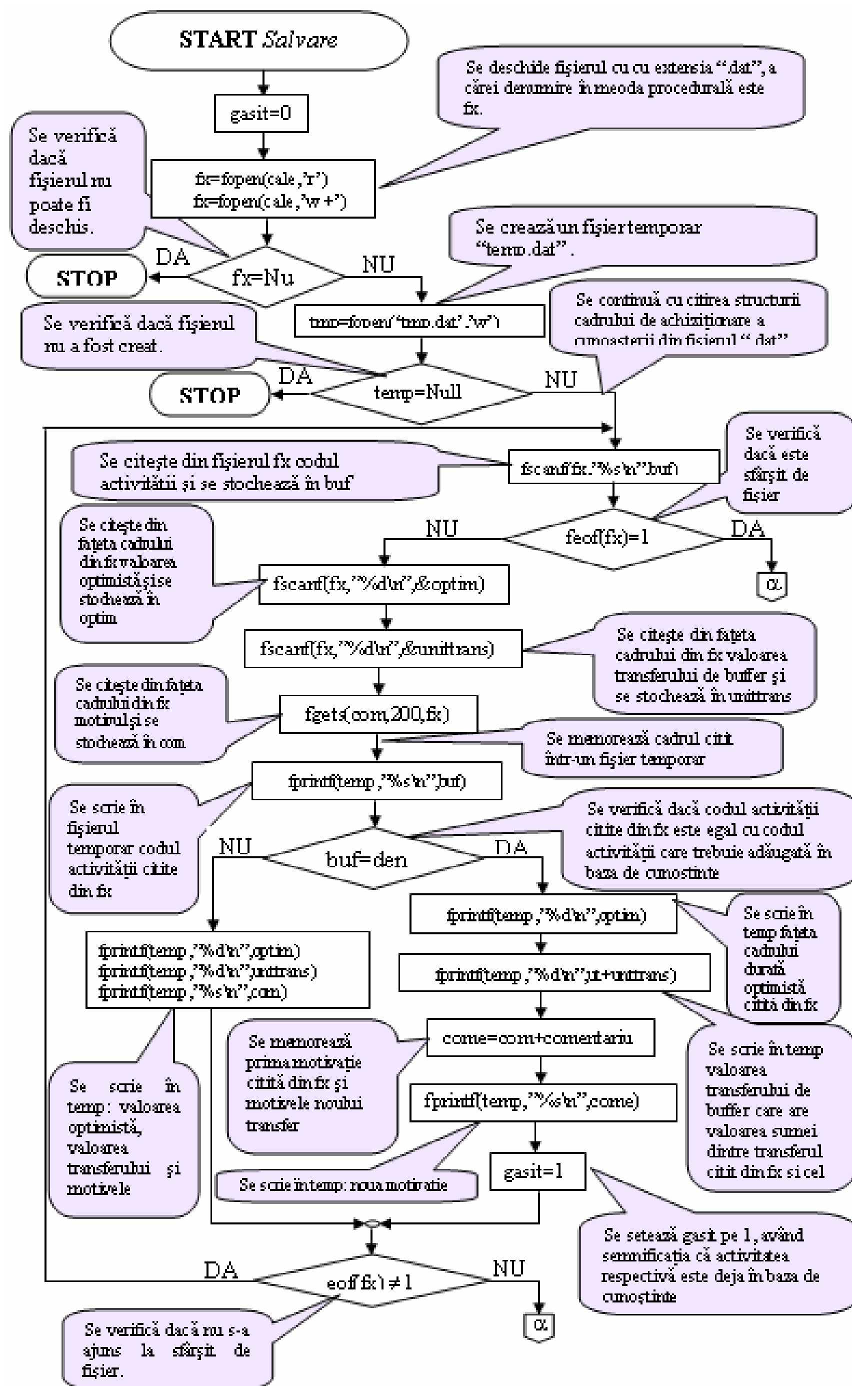


Fig. 3.12. Organigrama algoritmului "*Transfer de buffer*"


START Salvare
gasit=0
Se deschide fişierul cu cu extensia "dat", a cărei denumire în meoda procedurală este fx.
fx=fopen(cale,'r')
fx=fopen(cale,'w+')
Se verifică dacă fişierul nu poate fi deschis.
DA
STOP
fx=Nu
NU
Se crează un fişier temporar "temp.dat".
tmp=fopen('tmp.dat','w')
Se verifică dacă fişierul nu a fost creat.
DA
STOP
temp=Null
NU
Se continuă cu citirea structurii cadrului de achiziţionare a cunoasterii din fisierul "dat"
Se citeşte din fişierul fx codul activităţii şi se stochează în buf
fscanf(fx,"%s\n",buf)
Se verifică dacă este sfârşit de fişier
NU
feof(fx)=1
DA
α
Se citeşte din faţeta cadrului din fx valoarea optimistă şi se stochează în optim
fscanf(fx,"%d\n",&optim)
fscanf(fx,"%d\n",&unittrans)
Se citeşte din faţeta cadrului din fx valoarea transferului de buffer şi se stochează în unittrans
Se citeşte din faţeta cadrului din fx motivul şi se stochează în com
fgets(com,200,fx)
Se memorează cadrul citit într-un fişier temporar
fprintf(temp,"%s\n",buf)
Se scrie în fişierul temporar codul activităţii citite din fx
Se verifică dacă codul activităţii citite din fx este egal cu codul activităţii care trebuie adăugată în baza de cunostinte
NU
buf=den
DA
fprintf(temp,"%d\n",optim)
Se scrie în temp faţeta cadrului durată optimistă citită din fx
fprintf(temp,"%d\n",optim)
fprintf(temp,"%d\n",unttrans)
fprintf(temp,"%s\n",com)
fprintf(temp,"%d\n",ut+unttrans)
Se scrie în temp valoarea transferului de buffer care are valoarea sumei dintre transferul citit din fx si cel
Se memorează prima motivaţie citită din fx şi motivele noului transfer
come=com+comentariu
fprintf(temp,"%s\n",come)
Se scrie în temp: valoarea optimistă, valoarea transferului şi motivele
Se scrie în temp: noua motivatie
gasit=1
Se setează gasit pe 1, având semnificaţia că activitatea respectivă este deja în baza de cunoştinte
DA
eof(fx)≠1
NU
α
Se verifică dacă nu s-a ajuns la sfârşit de fişier.

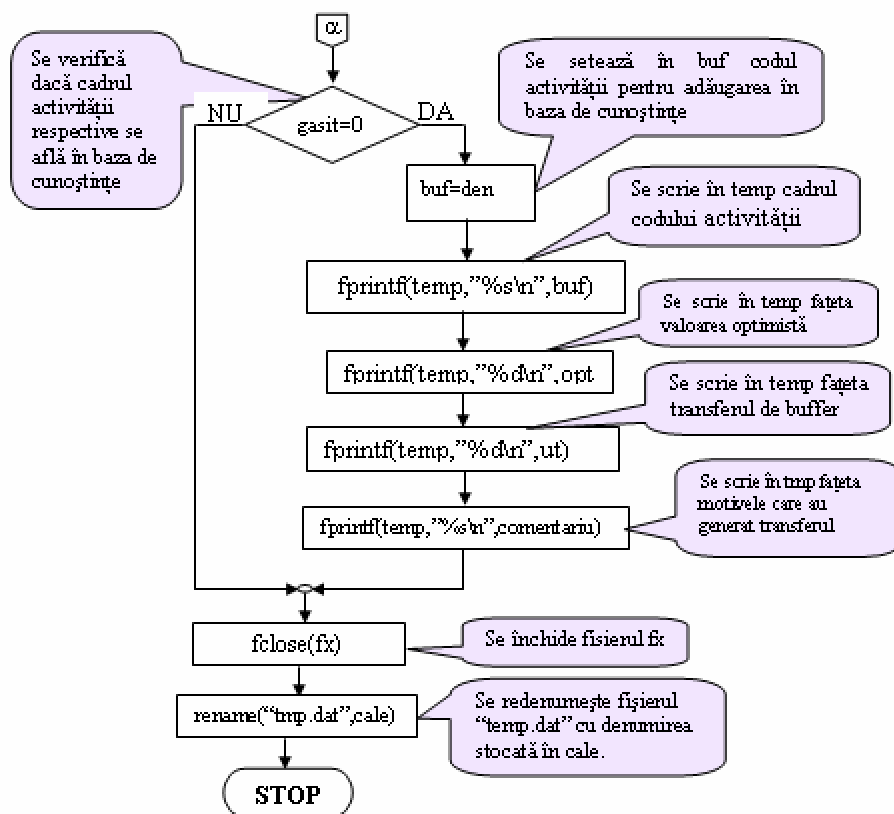


Fig. 3.13. Organigrama algoritmului "*Salvare*"

Se verifică dacă transferul cerut de buffer este pozitiv, (nu este posibil un transfer de buffer negativ deoarece proiectul evoluează conform valorii optimiste (strict operațională). Se compară ulterior valorea de timp cerută pentru tranfer cu valoare de timp stocată în buffer. În cazul unei valori de timp mai mici stocate în buffer decât cea necesară pentru transferul cerut, operația nu este posibilă. După ce s-au realizat verificările, se generează transferul de timp din buffer, care atrage după sine următoarele setări şi verificări:

- se memorează în fişierul cu extensia "**.dat**" codul activității, valoarea inițială a duratei optime şi valoarea transferului procesat.
- se verifică dacă valoare actualizată a duratei optimiste este mai mică decât cea a duratei pesimiste, în caz contrar se actualizează atât durata probabilă cât şi cea pesimistă, la noua valoare a duratei optimiste.(Fig. 3.12)

Procedura "*Transfer de Buffer*" include la rândul său procedura "*Salvare",* care a fost concepută pentru achiziționarea cunoaşterii structurate sub formă de cadre în baza de cunoştințe a sistemului (Fig.3.13).

4 *Analiza – diagnostic pentru activitățile proiectului*

a Prezentarea algoritmului

Algoritmul prin care se realizează *diagnosticul unei activități* are la bază arborele de căutare al Sistemului Expert **PManager** (§ 7.2.5, Fig.7.7)

Dacă o activitate este în progres atunci se calculează procentul de realizare al ei în cadrul proiectului. Pentru a stabili dacă o activitate este critică, se verifică dacă sursa activității şi destinația, sunt elemente consecutive ale vectorului **path**. În caz afirmativ rezultă că activitatea respectivă este critică. Pentru stabilirea cauzei care a plasat activitatea pe secvența critică se baleează toate transferurile cumulate care ar fi putut genera acest lucru.

Aceasta se realizează cu ajutorul procedurii recursive "*Path"* care identifică toate secvențele de activități care preced un anumit nod (eveniment), determinând drumul cel mai lung până la nodul sursă primit ca parametru. (Fig. 8.14)

Algoritmul este prezentat mai jos în pseudocod:

```
dacă sursa=0 atunci
        val=0
        Pentru j=0 la noduri
                        Nodurilanțuri[p][j]=lant[j]
                        val=val+vallanț[j]
        Sfârşit pentru
        Nodurilant[p][j]=0
        Valoaredr[p][0]=val
        P=p+1 ;
altfel
        Pentru i=0 la noduri
                        Dacă work[i][sursa]>0 atunci
                        lanț[nod]=sursa;
                        Vallanț[nod]=work[i][sursa]
                        nod=nod+1
                        Lant(i)
                        nod=nod-1
       Sfârşit pentru
```

Pentru a determina dacă o activitate a fost critică, se verifică dacă activitatea respectivă aparține drumului citit din fişierul „**crt**" şi dacă nodurile activității sunt noduri succesive în drumul respectiv.

Algoritmul care generează "*asistentul*" este următorul: se ia matricea **t** în care sunt stocate activitățile finalizate şi se determină care activitate a fost finalizată ultima şi toate activitățile care sunt în curs de desfăşurare. În continuare se calculează pentru fiecare activitate procentul de realizare conform planificatorului inițial şi se compară cu valoarea reală specificată de utilizator. În cazul unor valori diferite, algoritmul calculează transferul necesar de timp din *buffer* pentru o actualizare corectă a planificatorului conform sincronizării de timp.

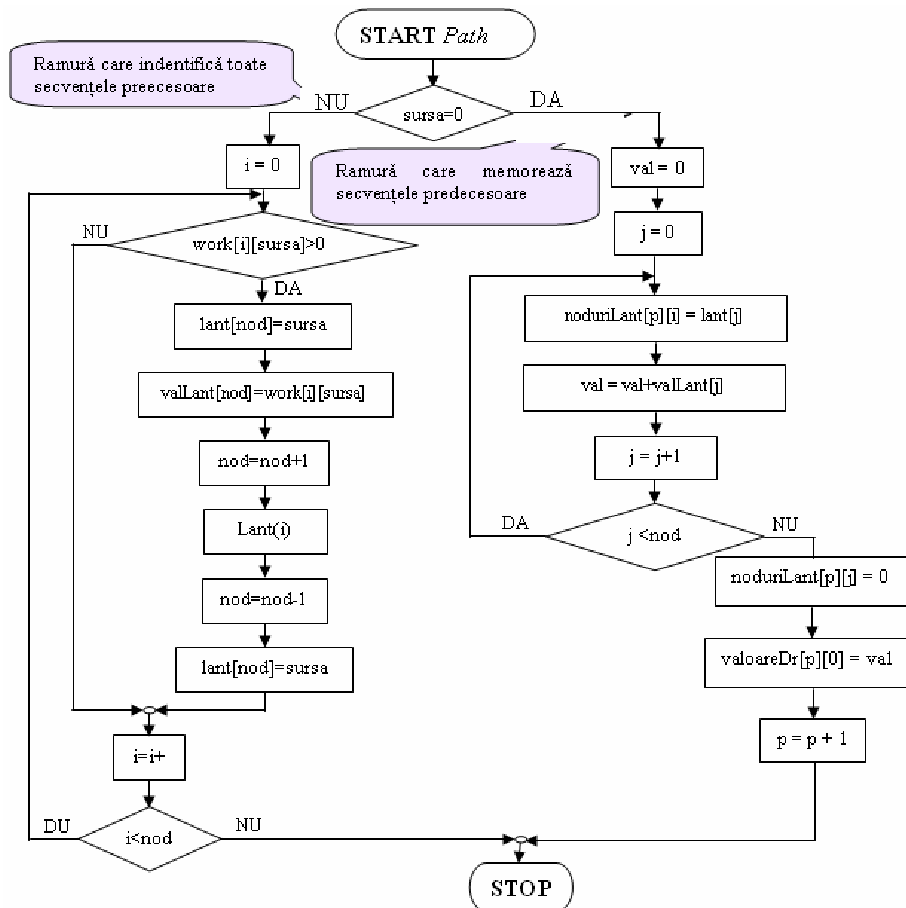


Fig. 3.14. Organigrama algoritmului "*Path*"

5 *Trasarea graficului pentru evoluți costurilor*

a Prezentarea algoritmului

Pentru a stabili coordonatele graficului de evoluție a costurilor sunt necesare să se creeze pe lângă matricea **work** încă trei matricii şi un vector.

- Matricea **cost** - este o matrice de adiacență a grafului cu deosebirea că în loc de valoarea optimistă se păstrează costul activității de la nodul **i** la nodul **j**.
- Matricea **costu** - este tot o matrice de adiacență în care se păstrează costul unitar al fiecărei activități.
- Cu ajutorul matricelor **dx** şi **t** calculate în cadrul procedurii de sincronizare se determină cea de-a treia matrice numită **costt**. În cadrul acestei matrici se setează ca număr de linii, numărul de activități în curs de desfăşurare, plus numărul de activități care sunt finalizate; numărul de coloane este dat de numărul de unități de timp parcurse de la startarea proiectului. Matricea **costt** este inițializată cu "0".

Matricele **dx** şi **t** sunt salvate într-o matrice **m**.

Pentru a se completa matricea **costt** se parcurg următorii paşi:

1. se iau activitățile finalizate şi în curs de desfăşurare pentru care se citeşte valoarea contorului **j** şi matricea **work,** având următoarele semnificații

- j este inițializat cu valoarea momentului de start al activității
- work[m[i][0]][m[i][1]] – conține durata activității în cauză

2. se completează matricea cost conform relației

- costt[i][j]= costu[m[i][0][m[i][1]] – reprezentând repartizarea unităților de cost pe unitățile de timp aferente activităților finalizate şi în curs de desfăşurare.

● Vectorul **costfinal** - este un vector format din costul cumulat pe fiecare unitate de timp.

În fig 3.15 este reprezentat algoritmul de calculul pentru cele trei costuri, CBMP, CRMP, CBMR (§ 4. 4) implementat în Sistemul Expert **PManager**.

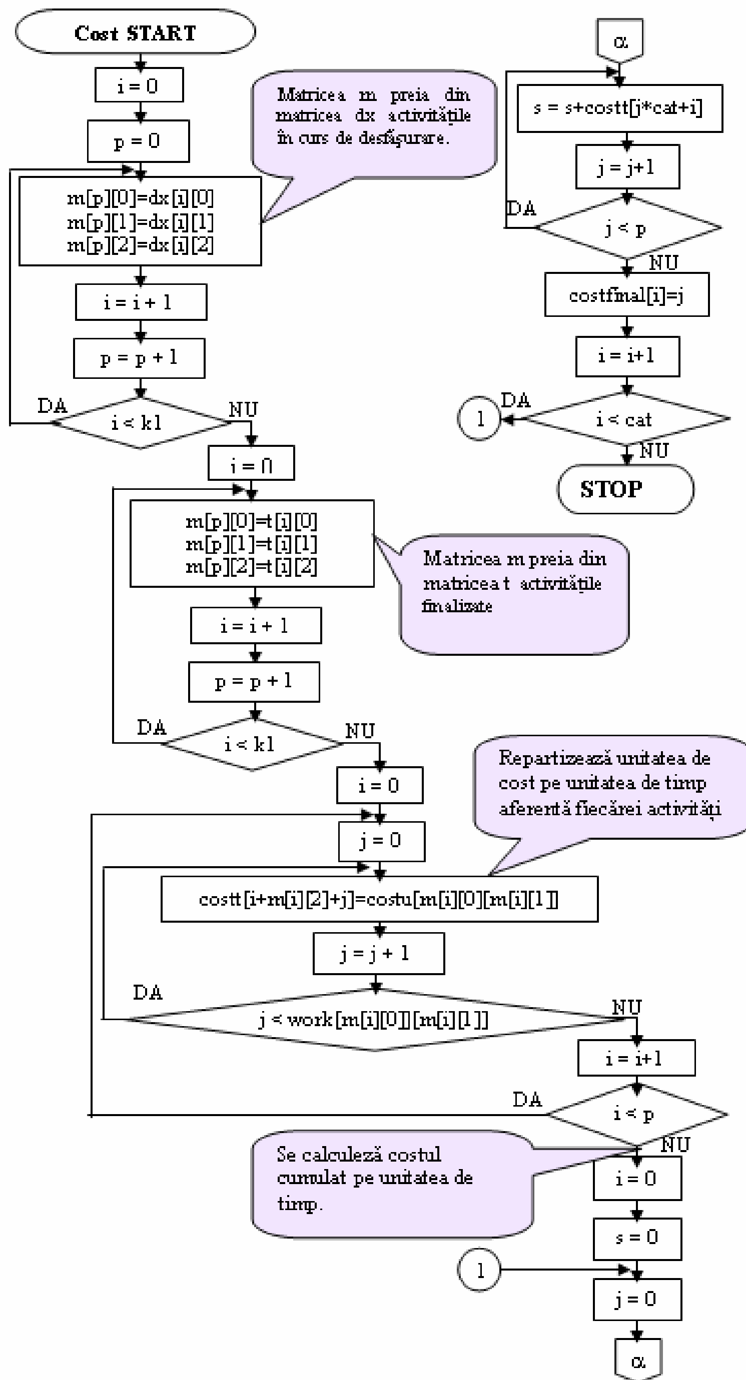


Fig. 3.15. …….

b

Strategia de control în spațiul stărilor pentru procedura de trasare a graficului evoluției costurilor

Coordonatele de trasare grafică a evoluției costurilor sunt cuprinse într-o matrice **coord**[i][4]. Ele se obțin parcurgând vectorul **cost** şi **coord**[i][0]=**coord**[i-1][2],

coord[i][1]=coord[i-1][3];

coord[i][2]=i+1;

coord[i][3]=coord[i][2]+costfinal[i].

Fie matricile:

$$\mathbf{work}=\begin{pmatrix}-1 & 3 & 4 & -1 & -1 & -1\\ -1 & -1 & -1 & -1 & -1 & -1\\ -1 & -1 & -1 & -1 & -1 & -1\\ -1 & 3 & -1 & -1 & -1 & -1\\ -1 & 5 & 3 & -1 & -1 & -1\\ -1 & -1 & 7 & 5 & 8 & -1\end{pmatrix};\ \mathbf{cost}=\begin{pmatrix}-1 & 6 & 12 & -1 & -1 & -1\\ -1 & -1 & -1 & -1 & -1 & -1\\ -1 & -1 & -1 & -1 & -1 & -1\\ -1 & 3 & -1 & -1 & -1 & -1\\ -1 & 10 & 12 & -1 & -1 & -1\\ -1 & -1 & 7 & 15 & 8 & -1\end{pmatrix};$$

Se calculează matricea costu[i][j]=cost[i][j]/work[i][j].

$$\mathbf{costu}=\begin{pmatrix}-1 & 2 & 3 & -1 & -1 & -1\\ -1 & -1 & -1 & -1 & -1 & -1\\ -1 & -1 & -1 & -1 & -1 & -1\\ -1 & 1 & -1 & -1 & -1 & -1\\ -1 & 2 & 4 & -1 & -1 & -1\\ -1 & -1 & 1 & 3 & 1 & -1\end{pmatrix};$$

Să presupune că s-au parcurs în cadrul proiectului 7 unități de timp. Matricea **dx** va arăta în acest caz astfel:

$$\mathbf{dx}=\begin{pmatrix}1 & 4 & 3\\ 2 & 5 & 4\end{pmatrix}\text{ iar matricea } \mathbf{t}=\begin{bmatrix}0 & 1 & 0\\ 0 & 2 & 0\\ 1 & 3 & 3\\ 2 & 4 & 4\end{bmatrix}.$$

Rezultă că matricea **m** va fi de forma:

$$\mathbf{m}=\begin{pmatrix}1 & 4 & 3\\ 2 & 5 & 4\\ 0 & 1 & 0\\ 0 & 2 & 0\\ 1 & 3 & 3\\ 2 & 4 & 4\end{pmatrix}$$

Parcurgând algoritmul menționat matricea **costt** va deveni:

$$\mathbf{costt}=\begin{pmatrix}2 & 2 & 2 & 0 & 0 & 0 & 0\\ 3 & 3 & 3 & 3 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 1 & 1 & 0\\ 0 & 0 & 0 & 2 & 2 & 2 & 2\\ 0 & 0 & 0 & 0 & 4 & 4 & 4\\ 0 & 0 & 0 & 0 & 1 & 1 & 1\end{pmatrix};$$

iar $\mathbf{costfinal}=\begin{bmatrix}5 & 5 & 5 & 6 & 8 & 8 & 7\end{bmatrix};$

$$\mathbf{coord} = \begin{bmatrix} 0 & 0 & 1 & 5 \\ 1 & 5 & 2 & 10 \\ 2 & 10 & 3 & 15 \\ 3 & 15 & 4 & 21 \\ 4 & 21 & 5 & 29 \\ 5 & 29 & 6 & 37 \\ 6 & 37 & 7 & 44 \end{bmatrix}$$

Unitatea de timp considerată în exemplificare a fost minutul. Pe măsura acumulării timpului şi depăşirii cuantumului de o oră, unitate de timp se transformă automat în ore. Prin analogie, se produce în continuare transformarea unității de timp în zile.