11. Grafuri ponderate ("Weighted Graphs")

- Adeseori, modelarea problemelor practice presupune utilizarea unor grafuri în care arcelor li se asociază **ponderi** care pot fi greutăți, costuri, valori, etc.

- Astfel de grafuri se mai numesc şi **grafuri ponderate** ("**weighted graphs**").

    - Spre exemplu, pe **harta traseelor aeriene** ale unei zone, arcele reprezintă rute de zbor iar ponderile distanțe sau prețuri.

    - Într-un **circuit electric** unde arcele reprezintă legături, lungimea sau costul acestora sunt de regulă utilizate ca ponderi.

    - Într-o activitate de **planificare în execuție a taskurilor**, ponderea poate reprezenta fie timpul, fie costul execuției unui task, fie timpul de aşteptare până la lansarea în execuție a taskului.

- Este evident faptul că într-un asemenea context apar în mod natural probleme legate de **minimizarea costurilor**.

- În cadrul acestui paragraf vor fi prezentate mai în detaliu două astfel de probleme referitoare la grafurile ponderate:

    - (1) Găsirea drumului cu costul cel mai redus care conectează toate punctele grafulu

    - (2) Găsirea drumului cu costul cel mai redus care leagă două puncte date.

- Prima problemă care este în mod evident utilă pentru grafuri reprezentând circuite electrice sau ceva analog, se numeşte **problema arborelui de acoperire minim** ("**minimum spanning tree problem**").

- Cea de-a doua problemă este utilă în grafurile reprezentând hărți de trasee (aeriene, feroviare, turistice) şi se **numeşte problema drumului minim** ("**shortest-path problem**").

- Aceste probleme sunt tipice pentru o largă categorie de aspecte ce apar în prelucrarea grafurilor ponderate.

- Se impune o precizare.

    - De regulă algoritmii utilizați presupun căutarea prin parcurgere a grafului, motiv pentru care în mod intuitiv ponderile sunt asociate cu distanțele.

    - Se spune de obicei "cel mai apropiat nod" cu sens ce poziționare a nodului.

- De fapt acest mod de a concepe lucrurile este valabil în contextul problemei drumului minim.

- În general, este însă foarte important a se avea în vedere faptul, că nu este absolut necesar ca ponderile să fie proporționale cu distanțele şi că ele pot reprezenta orice altceva, ca spre exemplu timpi, costuri sau valori.

    - În situațiile în care ponderile reprezintă într-adevăr distanțe, alți algoritmi cu caracter specific, pot fi mai potriviți decât cei care vor fi prezentați în continuare.

- În figura 11.a (a) apare o reprezentare grafică a unui graf neorientat ponderat.

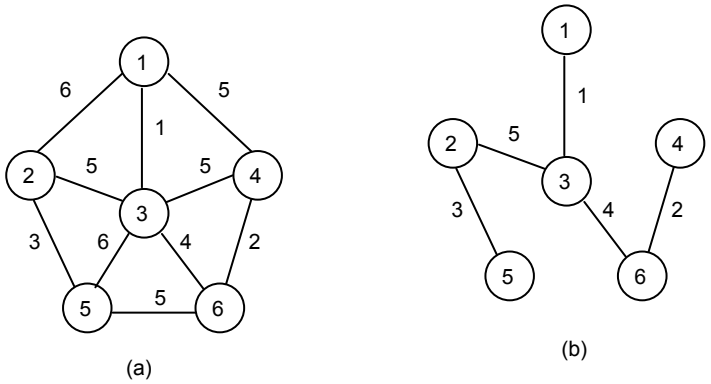

Fig.11.a Reprezentarea unui graf ponderat şi a unui arbore de acoperire minim asociat

- În ceea ce priveşte reprezentarea structurilor de date abstracte graf ponderat, principial ele se reprezintă ca şi grafurile normale cu următoarele deosebiri.

    - (1) În cazul reprezentării prin matrice de adiacențe, matricea va conține ponderi în locul valorilor booleene.

    - (2) În cazul reprezentării prin structuri de adiacențe, fiecărui element al listei i se adăugă un câmp suplimentar pentru memorarea ponderii.

- Se presupune faptul că ponderile sunt toate pozitive.

    - Există însă algoritmi mult mai complicați care pot trata şi ponderi negative.

    - Astfel de algoritmi sunt utilizați mai rar în activitatea practică.

11.1. Arbori de acoperire minimi ("Minimum-Cost Spanning Trees")

- Se presupune că `G=(N,A)` este un graf conex în care oricare arc `(u,v)` aparținând lui `A` are atașat un cost specific `cost(u,v)`.

- Un **arbore de acoperire** al lui `G` este un arbore liber care cuprinde toate nodurile din `N` (fig.11.1.(b)).

    - **Costul** unui arbore de acoperire este suma costurilor tuturor arcelor cuprinse în arbore.

- **Definiție**: Un **arbore de acoperire minim** al unui graf ponderat este o selecție de arce care conectează toate nodurile grafului, astfel încât costul său este cel puțin la fel de mic ca şi costul oricărui alt arbore de acoperire al grafului.

- O altă **definiție** este următoarea.

    - Dându-se orice partiționare a mulțimii nodurilor unui graf în două submulțimi, **arborele de acoperire minim** conține arcele cu ponderea cea mai mică care conectează un nod aparținând unei submulțimi cu un nod aparținând celeilalte [Se 88].

- Se face precizarea că un arbore de acoperire minim **nu** este în mod necesar unic.

- O aplicație tipică a arborilor de acoperire minimi o reprezintă proiectarea rețelelor de comunicații.

    - Nodurile grafului reprezintă oraşele iar arcurile comunicațiile posibile dintre ele.

    - Costul asociat unui arc reprezintă de fapt costul selecției acelei legături a rețelei.

    - Un arbore de acoperire minim reprezintă o rețea care conectează cu un cost minim toate oraşele.

11.1.1. Proprietatea arborilor de acoperire minimi

- Există mai multe moduri de a construi arbori de acoperire minimi asociați unui graf ponderat.

- Marea lor majoritate se bazează pe următoarea proprietate, denumită şi **proprietatea arborilor de acoperire minimi.**

    - Fie `G=(N,A)` un graf conex şi o funcție de cost definită pe arcele sale.

    - Fie `U` o submulțime proprie a mulțimii de noduri `N`.

    - Dacă `(u,v)` este un arc cu costul cel mai scăzut astfel încât `u∈U` iar `v∈N-U`, atunci există un **arbore de acoperire minim** care include pe `(u,v)` ca şi arc.

- Demonstrația acestei aserțiuni nu este complicată şi ea se realizează prin metoda reducerii la absurd.

    - Se presupune dimpotrivă că **nu** există un arbore de acoperire minim al lui `G` care include arcul `(u,v)`.

    - Fie `T` oricare arbore de acoperire minim al lui `G`.

- Adăugarea arcului `(u,v)` arborelui `T` trebuie să conducă la apariția unui ciclu, deoarece `T` este un arbore liber şi conform proprietății (b) dacă unui arbore liber i se adaugă un arc el devine un graf ciclic (& 10.1.).

    - Acest ciclu include arcul `(u,v)`.

- În consecință, în ciclul nou format, trebuie să existe un alt arc `(u',v')` al lui `T` astfel încât `u'∈N−U,` după cum rezultă din figura 11.1.1.a.

    - Dacă acest lucru **nu** ar fi adevărat, atunci în cadrul ciclului nu ar exista o altă posibilitate de a ajunge de la nodul `v` la nodul `u` decât reparcurgând arcul `(u,v)`.

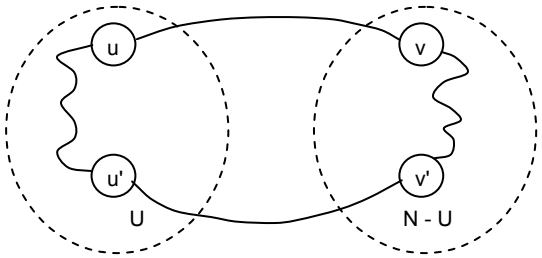

Fig.11.1.1.a. Demonstrarea proprietății arborilor de acoperire minimi

- Suprimând arcul `(u',v')`, ciclul dispare şi obținem arborele de acoperire `T'` al cărui cost **nu** este în siguranță mai ridicat decât al lui `T`, deoarece s-a presupus inițial că `cost(u,v)≤cost(u',v')`.

- Astfel existența lui `T'` contrazice presupunerea inițială şi anume că **nu** există un arbore de acoperire minim care să includă arcul `(u,v)` şi în consecință proprietatea arborilor de acoperire minimi este demonstrată.

11.2. Determinarea arborilor de acoperirie minimi

- Există mai multe metode de determinare a unui arbore minim asociat unui graf ponderat, metode care în general exploatează proprietatea anterior enunțată pentru aceşti arbori.

- Dintre acestea se remarcă cu deosebire:

    - (1) Algoritmul lui Prim

    - (2) Metoda căutării “bazate pe prioritate”

    - (3) Algoritmul lui Kruskal.

11.2.1. Algoritmul lui Prim

- Fie `G` graful ponderat pentru care se doreşte determinarea unui arbore de acoperire minim

- Fie N={1,2,3,…,n} mulțimea nodurilor grafului G

- Fie U o mulțime vidă de noduri ale grafului.

- **Algoritmul lui Prim** începe prin a introduce în mulțimea U nodul de pornire, să zicem nodul {1}.

  - În continuare, într-o manieră ciclică, este construit pas cu pas arborele de acoperire minim.

  - Astfel, în fiecare pas al algoritmului:

    - (1) Se selectează arcul cu cost minim (u,v) care conectează mulțimea U cu mulțimea N-U

    - (2) Se adaugă acest arc arborelui de acoperire minim

    - (3) Se adaugă nodul v lui U.

  - Ciclul se repetă până când U=N. Schița algoritmului lui Prim apare în secvența [11.2.1.a] .

```
{Construcția unui arbore de acoperire minim T al unui graf
G}

procedure PRIM(G: TipGraf; var T: MultimeDeArce);
var U: MultimeDeNoduri;
    u,v: TipNod;
  begin
    T:= Φ; U:= {nodul de pornire};
    while U <> N do                                  [11.2.1.a]
      begin
        *fie (u,v) arcul cu costul cel mai redus care
         satisface condiția (u IN U) AND (v IN N-U);
        T:= T ∪ {(u,v)}; U:= U ∪ {v}
      end
  end; {PRIM}
```

- În figura 11.2.1.a apare reprezentată pas cu pas secvența de construcție a arborelui de acoperire minim pentru graful (0) din aceeaşi figură.

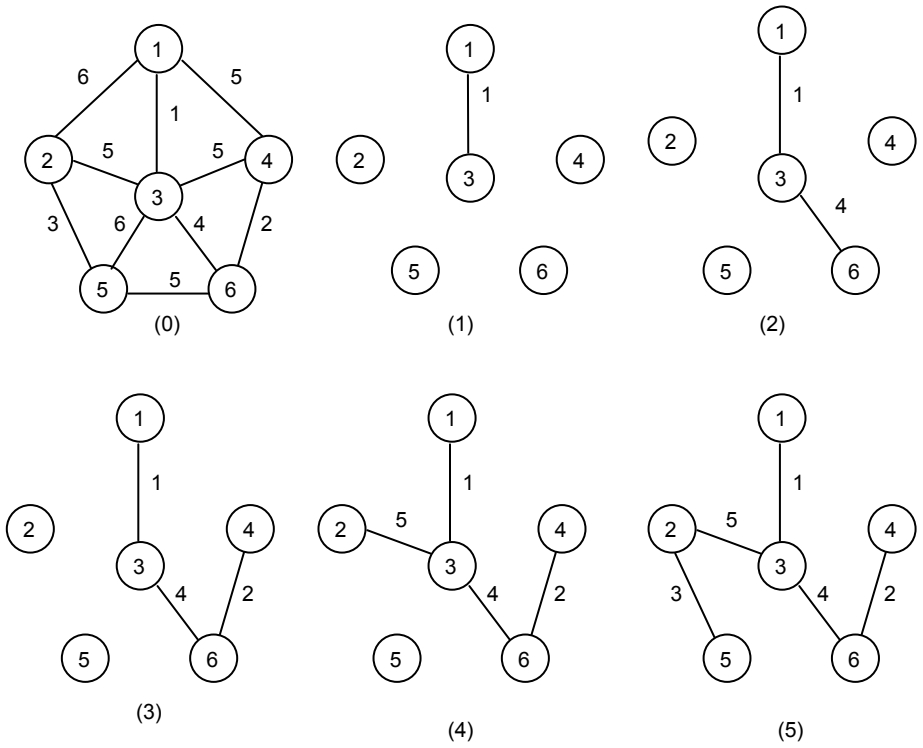

Fig.11.2.1.a. Construcția unui arbore de acoperire minim al unui graf pe baza algoritmului lui Prim

11.2.1.1. Exemplu de implementare a algoritmului lui Prim

- Se presupune un graf definit prin:
  - (1) Mulțimea nodurilor sale `{1,2,3,…,n}`
  - (2) Matricea `COST` care memorează costurile arcelor sale
- O modalitate simplă de a afla arcul cu costul cel mai redus care conectează mulțimile `U` şi `N-U`, este aceea de a utiliza două tablouri.
  - Unul dintre tablouri, denumit `APROPIAT` memorează în locația `i` acel nod din mulțimea `U` care este la momentul respectiv cel mai apropiat de nodul `i` din mulțimea `N-U`.
    - Acest tablou materializează de fapt mulțimea `N-U`.
  - Cel de-al doilea tablou denumit `COSTMIN,` memorează în locația `i` costul arcului `(i,APROPIAT[i])`.
- La fiecare pas al algoritmului se balează tabloul `COSTMIN` pentru a găsi acel nod, să-l denumim `k`, aparținând mulțimii `N-U,` care este cel mai apropiat de nodurile mulțimii `U`, adică nodul pentru care valoarea `COSTMIN[k]` este minimă.

- Se tipăreşte arcul (k,APROPIAT[k]) ca şi aparţinând arborelui de acoperire minim.

- În continuare se actualizează tablourile COSTMIN şi APROPIAT luând în considerare faptul că nodul k a fost adăugat mulţimii U. În consecinţă:

  - (1) Pe de o parte apar noi posibilităţi de conectare între cele două mulţimi

  - (2) Pe de altă parte costurile unor conexiuni existente se pot reduce prin introducerea noului nod k în mulţimea U.

- O versiune Pascal a algoritmului lui Prim apare în secvenţa [11.2.1.1.a].

- Se presupune că COST este un tablou de dimensiuni n,n astfel încât COST[i,j] reprezintă costul arcului (i,j).

  - Dacă arcul (i,j) nu există, COST[i,j] are o valoare mare specifică.

- Ori de câte ori se găseşte un nod k pentru a fi introdus în arborele de acoperire (mulţimea U), se face COSTMIN[k] egal cu valoarea "infinit".

  - Valoarea infinit reprezintă o valoare mare convenită, astfel încât acest nod să nu mai poată fi selectat în continuare spre a fi inclus în U.

  - Se face precizarea că valoarea "infinit" trebuie să fie mai mare decât costul oricărui arc al grafului, respectiv mai mare decât costul asociat lipsei arcului.

```
{Implementarea algoritmului lui Prim}

procedure Prim(COST: array[1..n,1..n] of real);

{afisează arcele arborelui de acoperire minim pentru un graf
având nodurile{1,2,...,n} şi matricea COST pentru costurile
arcelor}

var COSTMIN: array[1..n] of real;
    APROPIAT: array[1..n] of integer;
    i,j,k,min: integer;

{i şi j sunt indici. În timpul parcurgerii tabloului
COSTMIN, k este indicele celui mai apropiat nod găsit pâna
la momentul curent, iar min=COSTMIN[k]}

  begin
    for i:= 2 to n do
      begin {iniţializează mulţimea U numai cu nodul 1}
        COSTMIN[i]:= COST[1,i];
        APROPIAT[i]:= 1;
      end;                                          [11.2.1.1.a]
    for i := 2 to n do
      begin  {caută cel mai apropiat nod k din afara lui U,
              faţă de U}
        min := COSTMIN[2];
```

```
        k := 2;
        for j := 3 to n do
          if COSTMIN[j]< min then
            begin
              min:= COSTMIN[j]
              k:= j
            end;
        writeln(k,APROPIAT[k]); {tipăreşte arcul}
        COSTMIN[k]:= infinit; {k se adaugă lui U}
        for j:= 2 to n do {ajustează costurile lui U}
          if(COST[k,j]<COSTMIN[j]and COSTMIN[j]<infinit)
           then
            begin
              COSTMIN[j]:= COST[k,j];
              APROPIAT[j]:= k
            end
      end
  end; {Prim}
```

- În figura 11.2.1.1.b apare urma execuției algoritmului lui Prim aplicat grafului din figura 11.2.1.1.a.(0).

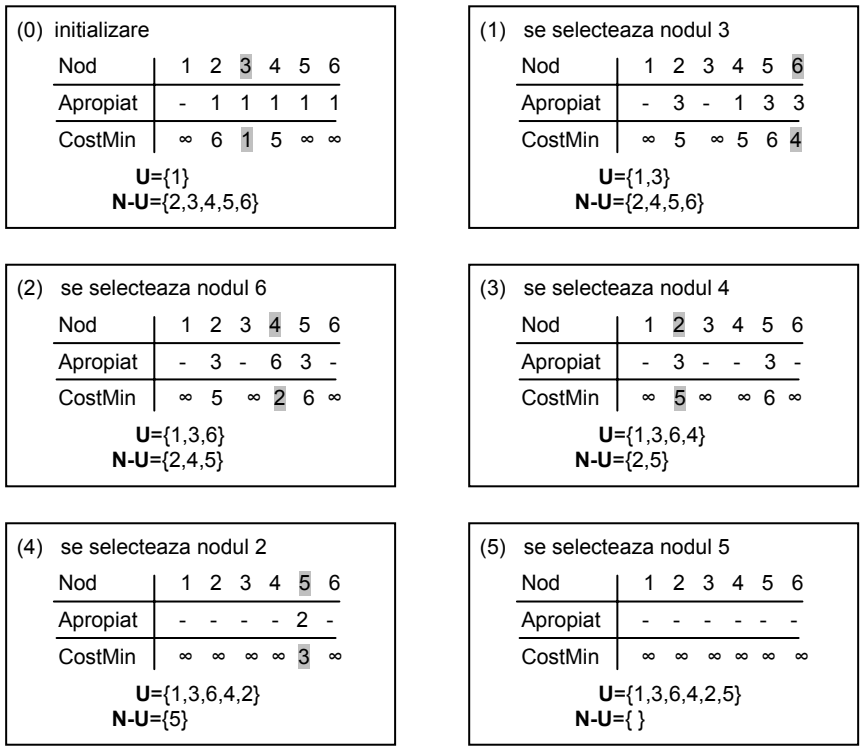

Fig.11.2.1.1.b. Exemplu de aplicare al algoritumului lui Prim

- Complexitatea relativă la timpul de execuție a acestei implementări a algoritmului lui Prim este $O(n^2)$, deoarece:
  - (1) Se fac `n-1` iterații în bucla exterioară **`for`**
  - (2) Fiecare iterație necesită $O(n)$ unități de timp datorită celor două bucle **`for`** interioare succesive, prima care determină arcul minim şi cea de-a doua care ajustează costurile lui `U`.
- Pentru valori mari ale lui `n`, performanța algoritmului poate deveni nesatisfăcătoare.

11.2.2. Metoda căutării "bazate pe prioritate" ("Priority-First Search")

- După cum s-a precizat în &10.4.3, considerând nodurile unui graf divizate în trei clase, "arbore", "vecinătate" şi "neîntâlnite", atunci metodele de traversare a grafului se diferenţiază după maniera în care sunt alese nodurile care trec din clasa "vecinătate" în clasa "arbore".

    - Astfel la traversarea "**în adâncime**" se alege din vecinătate nodul cel mai recent întânit (ultimul) ceea ce corespunde utilizării unei stive în păstrarea nodurilor clasei "vecinătate".

    - La traversarea "**prin cuprindere**" se alege nodul cel mai devreme întânit (primul) ceea ce corespunde unei structuri de date coadă.

- Determinarea arborelui de acoperire minim se poate asimila cu acea traversare a grafului în care se alege din clasa "vecinătate" nodul cu **prioritatea maximă**, adică acela la care conduce arcul cu ponderea minimă.

- Structura de date care poate fi asociată acestei metode este **coada bazată pe prioritate**.

- În secvenţa [11.2.2.a] se prezintă o metodă de determinare a arborelui minim bazată pe considerentele mai sus precizate.

    - Tehnica utilizată se mai numeşte şi **căutare bazată pe prioritate** ("priority first search") [Se 88].

- Referitor la această secvenţă se fac următoarele precizări:

1) Graful se consideră reprezentat printr-o **structură de adiacenţe**, implementată cu ajutorul listelor înlănţuite simple (&10.3.2, Caz 1).

2) Structura unui nod al unei liste de adiacenţe se completează cu câmpul "`cost`" utilizat pe post de prioritate.

3) Procedura ***`Initializeaza`*** şi funcţiile ***`Extrage`*** şi ***`Vid`*** implementează operatorii respectivi în contextul cozilor bazate pe prioritate.

4) Funcţia ***`Actualizeaza`***`(q:CoadaBazataPePrioritate, k: TipNod,p:TipPrioritate):boolean` implementează un operator nou care are menirea de a verifica dacă nodul `k` precizat ca parametru apare în coada bazată pe prioritate `q`, cu o prioritate cel puţin egală cu prioritatea `p` precizată ca parametru.

- (1) Dacă nodul `k` nu apare în coadă el este inserat.

- (2) Dacă nodul `k` apare în coadă însă are un cost mai mare (adică o prioritate mai mică) decât cea precizată ca parametru se realizează schimbarea priorităţii sale la valoarea `p`.

- (3) Dacă se produce vreo modificare (inserţie sau modificare de prioritate) funcţia ***`Actualizeaza`*** returnează valoarea "true". Aceasta permite actualizarea corespunzătoare a tablourilor `marc` şi `parinte`.

```
-------------------------------------------------------------
PROCEDURE ArboreMinim;

  VAR id,k: integer;
      marc: ARRAY[1..maxN] OF integer;
    parinte: ARRAY[1..maxN] OF integer;
    q: CoadaBazataPePrioritate;

  PROCEDURE CautaPrioritar(k:integer);
    VAR t: legatura;

    BEGIN
[1]   IF Actualizeaza(q,k,nevazut) THEN parinte[k]:= 0;
[2]   REPEAT
[3]     id:= id + 1;
[4]     k:= Extrage(q);
[5]     marc[k]:= -marc[k]; {k trece în clasa "arbore"}
[6]     IF marc[k] = nevazut THEN marc[k]:= 0;
[7]     t := Stradj[k]
[8]     WHILE t <> nil DO                          [11.2.2.a]
          BEGIN
[9]         IF marc[t^.nume]<0 THEN {nevizitat sau in coada}
[10]          IF Actualizeaza(q,t^.nume,t^.cost) THEN
                BEGIN
[11]              marc[t^.nume]:= -(t^.cost); {nodul t^.nume
                                    trece în clasa "vecinatate"}
[12]              parinte[t^.nume]:= k
                END;
[13]          t := t^.urm
          END
[14]  UNTIL Vid(q)
    END; {CautaPrioritar}

  BEGIN
    id := 0;Initializeaza(q);
    FOR k:= 1 TO N DO marc[k]:= -nevazut;
    FOR k:= 1 TO N DO
      IF marc[k] = -nevazut THEN CautaPrioritar(k)
  END; {ArboreMinim}
-------------------------------------------------------------
```

- Arborele de acoperire minim care în acest caz este un arbore de căutare bazată pe prioritate, este păstrat în tabloul `parinte` în reprezentarea "indicator spre părinte".

- Fiecare locație a tabloului memorează părintele nodului în cauză respectiv nodul care a determinat mutarea respectivului nod din clasa "vecinătate" în clasa "arbore".

- De altfel, pentru fiecare nod `k` al arborelui, `marc[k]` reprezintă costul arcului care-l leagă pe `k` de părintele său `părinte[k]`.

    - (1) Nodurile din clasa "arbore" sunt marcate cu valori pozitive în tabloul `marc`

    - (2) Nodurile din clasa "vecinătate" sunt marcate cu valori negative (linia [11])

    - (3) Nodurile din clasa "neîntâlnite" sunt marcate cu " – nevăzut" şi nu cu valoarea zero.

- Se face precizarea că "nevăzut" reprezintă o valoare mare pozitivă.

- Se observă faptul că atâta vreme cât nodurile se găsesc în coada bazată pe prioritate ele sunt marcate în tabloul `marc` cu valoarea negativă a costului (priorității).

- În momentul în care sunt trecute în clasa "arbore" li se inversează valoarea în tabloul `marc` (linia `[5]`).

- Utilizarea căutării bazate pe prioritate în determinarea arborelui de acoperire minim conduce la performanța O*((n+a) log n)* paşi de execuție.

- Motivația este simplă:

  - În procesul construcției arborelui sunt parcurse toate nodurile şi toate arcele.

  - Fiecare nod conduce la o inserție şi fiecare arc la o modificare de prioritate în cadrul cozii bazate pe prioritate utilizate.

  - Presupunând că implementarea cozii bazate pe prioritate s-a realizat cu ajutorul ansamblelor, atunci atât inserția cât şi modificarea se realizează în O*(log n)* paşi.

  - În consecință performanța totală va fi O*((n+a) log n).*

11.2.2.1. Considerații referitoare la metoda căutării "bazate pe prioritate"

- Metoda căutării “bazate pe prioritate” are un pronunțat caracter de **generalitate**.

- Astfel, după cum se va vedea în continuare, pornind de la această metodă se poate dezvolta un algoritm care rezolvă **problema drumului minim**.

- De asemenea, pornind tot de la aceeaşi metodă va fi dezvoltat un algoritm care rezolvă aceleaşi probleme în cazul grafurilor dense cu un efort de calcul proporțional cu O (n^2).

- Istoric lucrările au evoluat însă altfel.

  - În anul 1957 **Prim** publică algoritmul pentru determinarea arborelui de acoperire minim

  - În anul 1959 **Dijkstra** publică algoritmul referitor la determinarea drumului minim.

  - Pentru clarificare, se acceptă ca în cazul grafurilor dense cele două soluții să fie numite:

    - **Algoritmul lui Prim** pentru determinarea arborelui de acoperire minim al unui graf

    - **Algoritmul lui Dijkstra** pentru determinarea drumului minim într-un graf

        - **Algoritm de căutare bazată pe priorități** în cazul grafurilor rare.

- După cum se va vedea, de fapt soluțiile propuse se întrepătrund şi ele nu sunt decât cazuri particulare ale unui algoritm generalizat de căutare bazat pe priorități.

- Este evident faptul că metoda "căutării bazate pe prioritate" este aplicabilă cu preponderență în cazul grafurilor ponderate considerând ponderile drept priorități.

- Cu toate acestea, această metodă poate fi aplicată şi grafurilor neponderate.

- Într-un astfel de context, ea poate generaliza tehnicile de traversare bazate pe căutarea "în adâncime" respectiv pe căutarea "prin cuprindere" prin atribuirea corespunzătoare de priorități nodurilor grafului.

    - Se reaminteşte faptul că `id` este o variabilă a cărei valoare se incrementează de la `1` la `n` pe măsură ce sunt procesate nodurile grafului în timpul execuției procedurii `CautaPrioritar` din secvența [11.2.2.a]

    - Variabila `id` poate fi utilizată în atribuirea de priorități nodurilor examinate, în baza convenției "minim = prioritar".

    - Astfel, dacă în procesul de căutare bazată pe prioritate:

        - (1) Se consideră prioritatea unui nod egală cu `n-id` se obține căutarea "**în adâncime**"

        - (2) Dacă se consideră prioritatea unui nod se consideră egală chiar cu `id`, se obține căutarea "**prin cuprindere**".

    - În primul caz nodurile nou întâlnite au cea mai mare prioritate

    - În cel de-al doilea caz nodurile cele mai vechi adică cel mai devreme întâlnite, au cea mai mare prioritate.

    - De fapt aceste atribuiri de priorități determină structura **coadă bazată pe prioritate** să se comporte ca o **stivă** respectiv ca şi o **coadă normală**, structuri specifice celor două tipuri de parcurgeri.

11.2.3. Algoritmul lui Kruskal

- Fie graful conex `G=(N,A)` cu `N={1,2,...,n}` şi cu funcția de cost `c` definită pe mulțimea arcelor `A`.

- O altă metodă de construcție a unui arbore de acoperire minim, este **algoritmul lui Kruskal**

    - Algoritmul lui Kruskal porneşte de la un graf `T=(N,Φ)` care constă doar din cele `n` noduri ale grafului original `G`, dar care nu are nici un arc.

        - În această situație fiecare nod este de fapt o componentă conexă a grafului care constă chiar din nodul respectiv

- În continuare pornind de la mulțimea curentă a componentelor conexe, algoritmul selectează pe rând câte un arc de cost minim pe care îl adaugă componentelor conexe care cresc în dimensiune dar al căror număr se reduce.

- În final rezultă o singură componentă conexă care este chiar arborele de acoperire minim.

- Pentru a construi componente din ce în ce mai mari se examinează arcele din mulțimea `A` în ordinea crescătoare a costului lor.

    - Dacă arcul selectat conectează două noduri aparținând unor componente conexe distincte, arcul respectiv este adăugat grafului `T`.

    - Dacă arcul selectat conectează două noduri aparținând unei aceleeaşi componente conexe, arcul este neglijat întrucât introducerea sa ar conduce la apariția unui ciclu în respectiva componentă şi în final la un ciclu în arborele de acoperire, lucru nepermis prin definiție.

    - Aplicând în manieră repetitivă acest procedeu, la momentul la care toate nodurile grafului aparțin unei singure componente conexe, algoritmul se termină şi `T` reprezintă arborele de acoperire minim al grafului `G`.

- Cu alte cuvinte algoritmul lui Kruskal porneşte de la o pădure cu `n` arbori.

    - În fiecare din cei `n-1` paşi, algoritmul combină doi arbori într-unul singur, utilizând ca legătură arcul cu costul cel mai redus curent.

    - Procedeul continuă până în momentul în care rămâne un singur arbore.

    - Acesta este **arborele de acoperire minim**.

- Spre exemplu considerând graful ponderat din fig.11.2.3.a (0), în succesiunea (1)-(5) din cadrul aceleaşi figuri se prezintă maniera de determinare a unui arbore de acoperire minim al grafului în baza acestui algoritm.

    - Ordinea în care se adaugă arcele rezultă din figură.

    - Inițial se adaugă arcele cu costurile `1,2,3` şi `4`, toate acceptate, întrucât nici unul dintre ele nu generează vreun ciclu.

    - Arcele `(1,4)` şi `(3,4)` de cost `5` **nu** pot fi acceptate deoarece conectează noduri aparținând unei aceleeaşi componente (fig.11.2.3.a (d)) şi conduc la cicluri.

    - În consecință se acceptă arcul `(2,3)` de cost `5` care nu produce nici un ciclu încheind astfel construcția arborelui de acoperire minim.

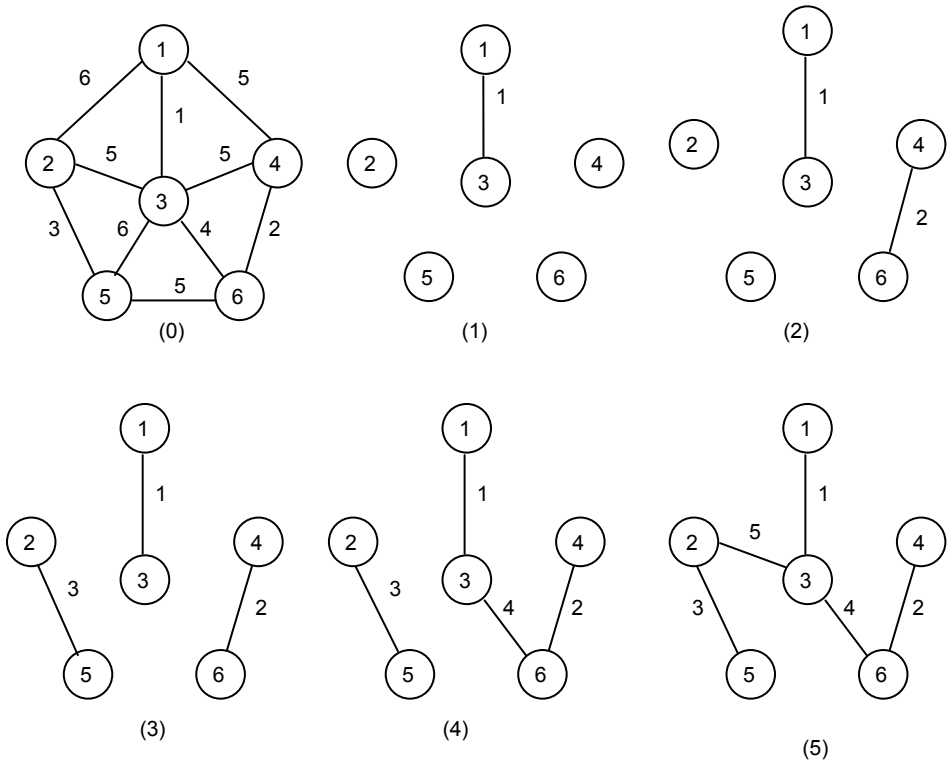

Fig.11.2.3.a. Construcția unui arbore de acoperire minim pe baza algoritmului lui Kruskal