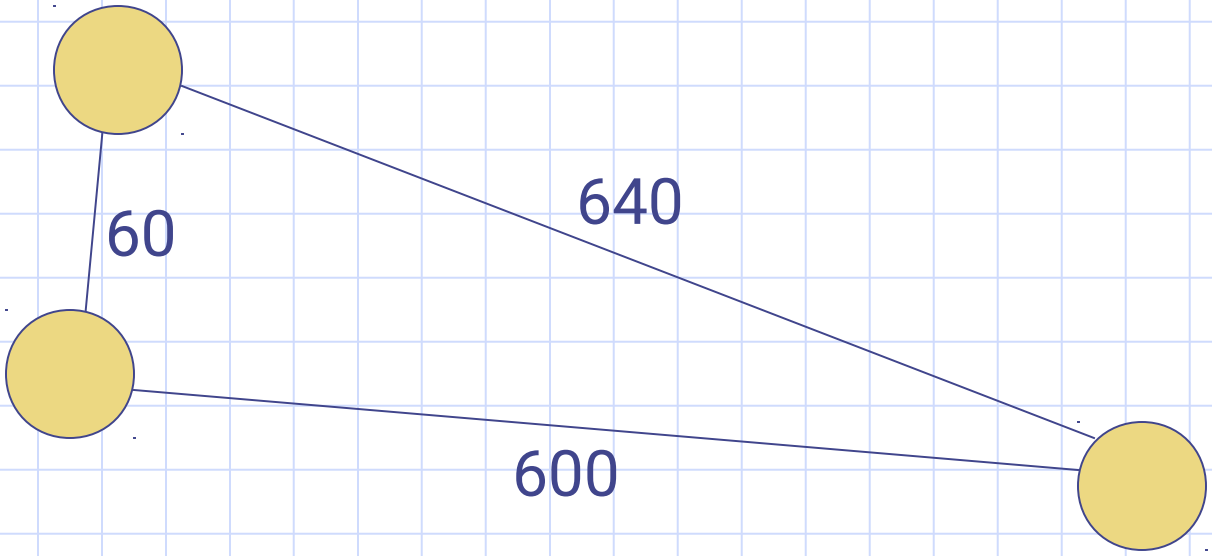
Algoritmul lui Prim

- Este un algoritm care returneaza un arbore de acoperire minim pentru un graf ponderat (graf in care fiecare arc are asociat un cost)
- Un arbore de acoperire pentru un graf este un subgraf alcatuit din toate nodurile grafului initial dar nu din toate arcele, ci doar din atatea arce cat sa nu apara cicluri (altfel nu ar fi arbore)

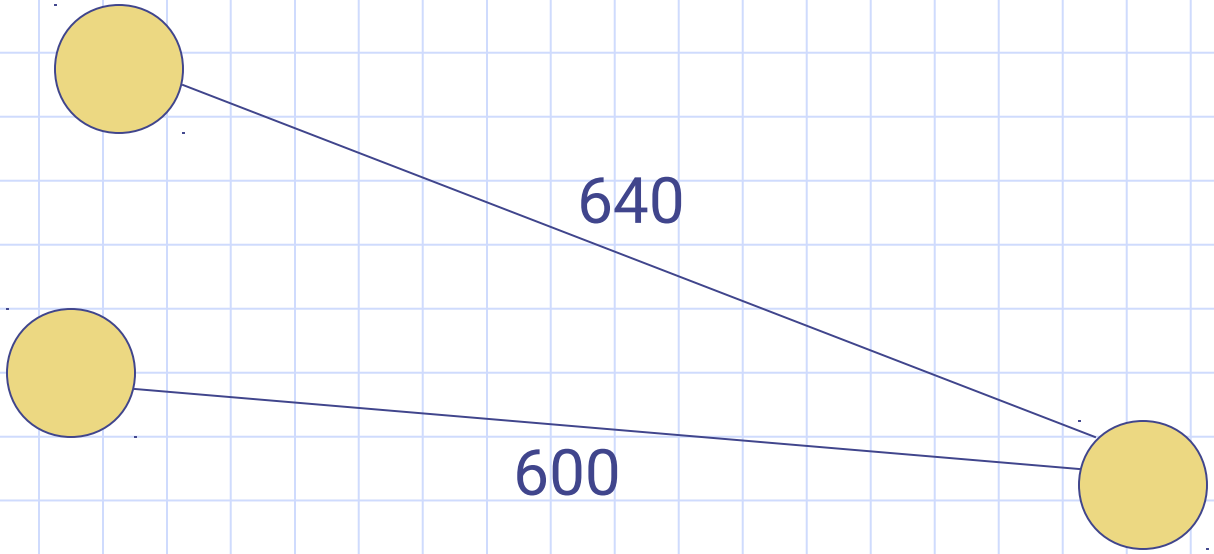
Algoritmul lui Prim

- Trebuie sa conectam 3 orase la o retea telefonica: Bucuresti, Timisoara si Arad
- Necesar cablu: 1300 km



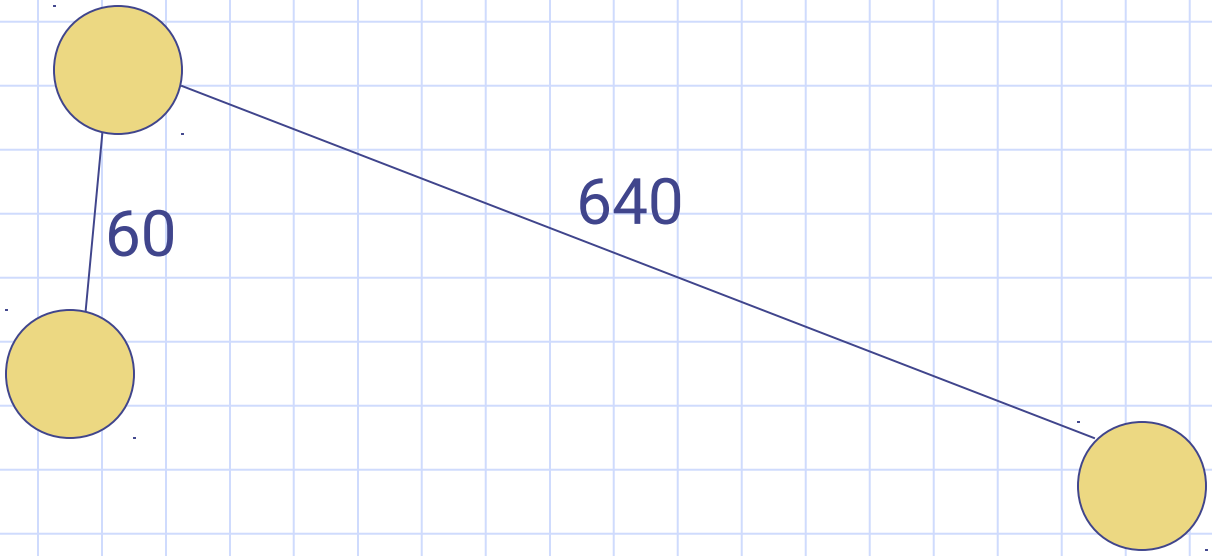
Algoritmul lui Prim

- E inutil sa executam toate cele trei conexiuni, numai doua din ele sunt suficiente pentru o comunicare in bune conditii intre oricare 2 orase
- De exemplu, legatura Timisoara – Arad ar putea lipsi, caz in care necesarul de cablu devine 1240 km



Algoritmul lui Prim

- Sau legatura Timisoara – Bucuresti ar putea lipsi, necesarul de cablu devenind 700 km

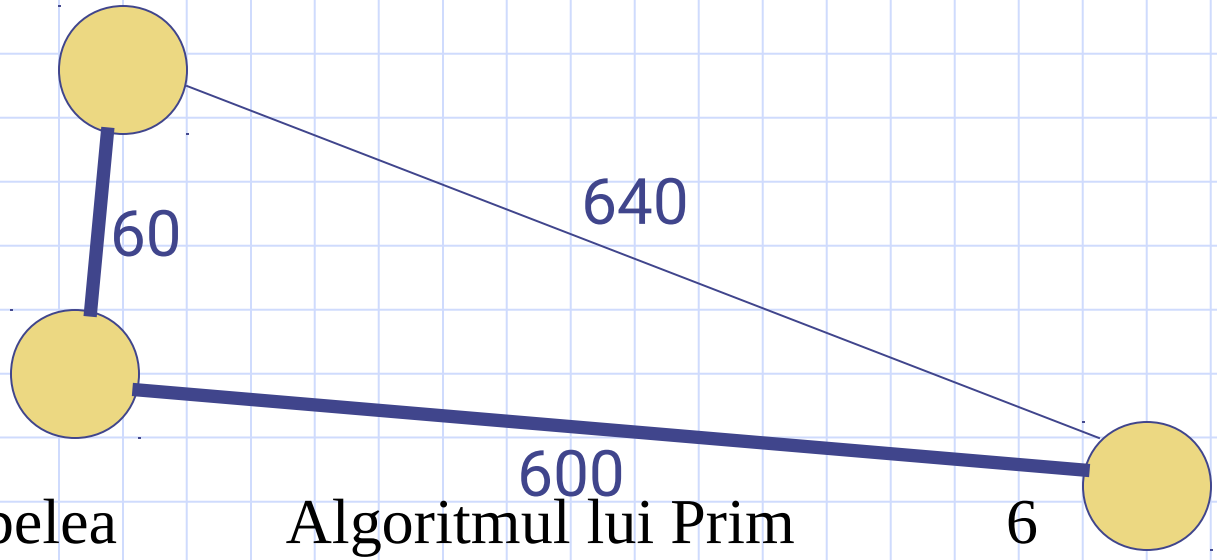


Algoritmul lui Prim

- Oricare 2 legaturi sunt suficiente, deoarece semnalul electric circula suficient de rapid ca un abonat din Timisoara care doreste sa vorbeasca cu unul din Arad (de exemplu) sa nu-si dea seama ca nu exista legatura directa intre Timisoara si Arad si ca apelul sau este rutat prin Bucuresti
- Din punctul de vedere al necesarului de cablu, lucrurile nu mai stau la fel
- Conteaza foarte mult care legaturi vor fi realizate si care nu
- Cel mai ieftin ar fi sa alegem legaturile Arad – Timisoara si Timisoara – Bucuresti si sa evitam legatura Arad - Bucuresti, necesarul de cablu ajungand in acest caz la 660 km; aceasta este situatia optima – sau “acoperirea minima” a retelei

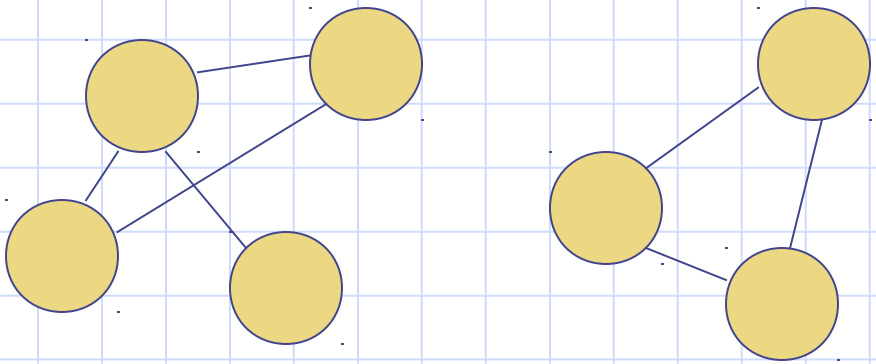
Algoritmul lui Prim

- Se observa ca trebuie determinat un arbore de acoperire pentru graful initial, adica un subgraf continand toate nodurile grafului insa doar atatea arce cat sa ramana un arbore (trebuie evitate ciclurile)
- Pentru un graf conex cu N noduri, un arbore de acoperire va avea N-1 arce (2 arce in cazul grafului Arad – Timisoara – Bucuresti discutat)



Algoritmul lui Prim

- Cazurile cele mai simple sunt cele ale grafurilor conexe, adica acelea in care din orice nod se poate ajunge in orice alt nod
- In figura de mai jos este prezentat un graf neconex alcatuit din 2 componente conexe, care n-au legatura una cu alta
- Cu alte cuvinte, in graful de mai jos nu se poate ajunge din orice nod in orice alt nod (daca nodurile sunt in componente conexe diferite)



Algoritmul lui Prim

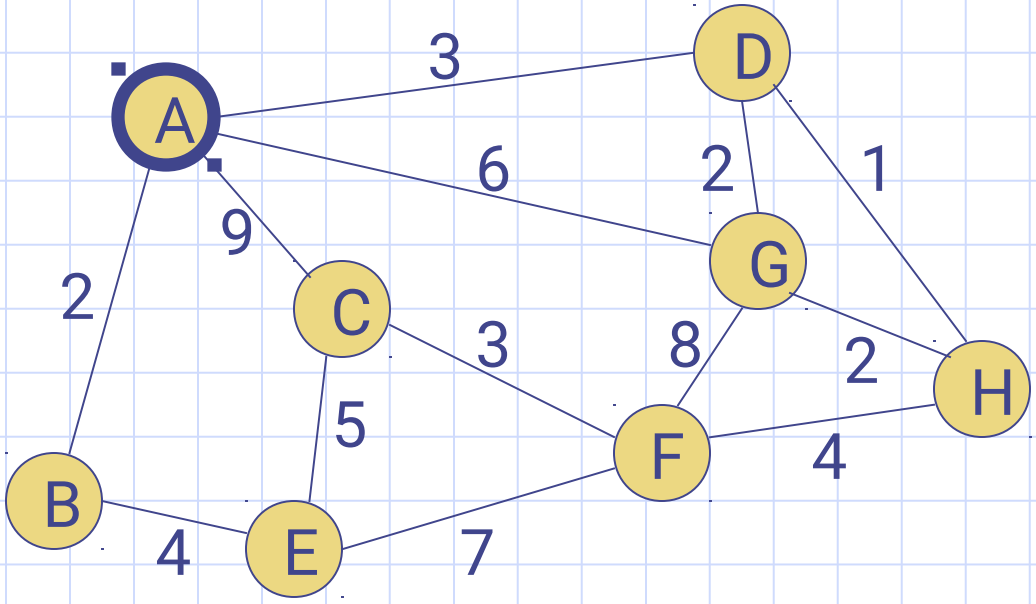
- Pentru un graf conex pot fi gasiti mai multi arbori de acoperire, in functie de arcele care sunt alese pentru a forma arborele
- Costul total al arborelui de acoperire este dat de suma costurilor arcelor alese, deci vom avea arbori “mai scumpi” si arbori “mai ieftini”
- Algoritmul lui Prim gaseste arborele de acoperire cel mai ieftin pentru un graf conex ponderat, pe care-l vom numi “arborele de acoperire minim” (acesta poate sa nu fie unic)
- Daca graful nu este conex, el este alcatuit din subgrafuri (componente) conexe
- In cazul unui astfel de graf, algoritmul lui Prim gaseste cate un arbore de acoperire minim pentru fiecare componenta conexa a grafului (neconex) dat, adica o “padure de arbori de acoperire minimi”

Algoritmul lui Prim

- Initial, toate nodurile grafului se considera nevizitate
- Se porneste de la ***un nod oarecare*** al grafului care se marcheaza ca vizitat
- In permanenta vor fi mentinute doua multimi:
  - Multimea U a nodurilor vizitate (initial, U va contine doar nodul de start)
  - Multimea N\U a nodurilor nevizitate (N este multimea tuturor nodurilor)
- La fiecare pas se alege acel nod din multimea N\U care este legat ***prin arc de cost minim*** de ***oricare*** din nodurile din multimea U
- Nodul ales va fi scos din multimea N\U si trecut in multimea U
- Algoritmul continua pana cand U = N

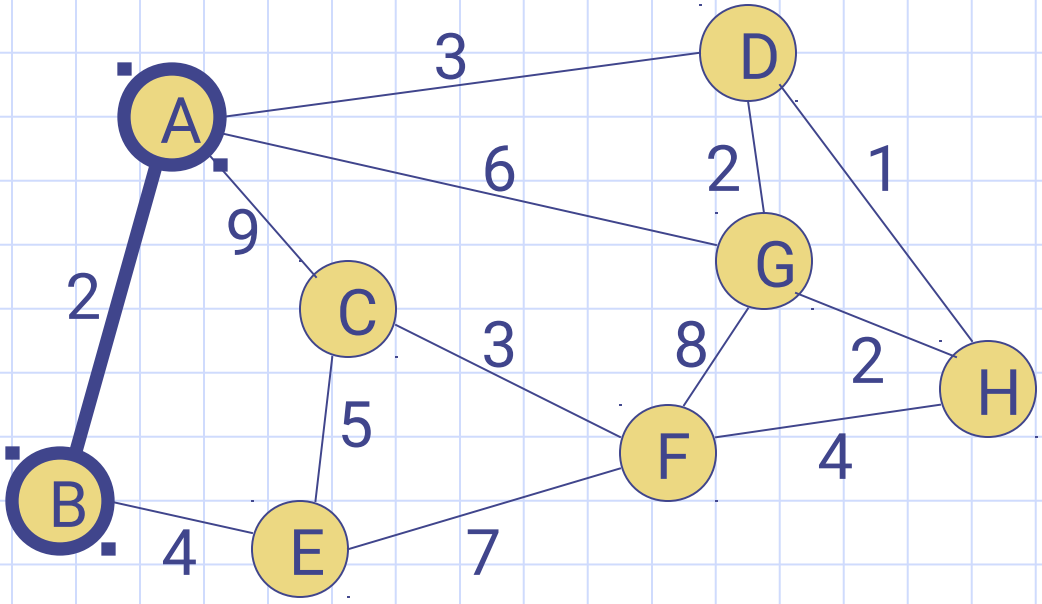
Algoritmul lui Prim

- Fie graful din figura
- Pornim de la nodul A (reusita algoritmului nu depinde de nodul de start)



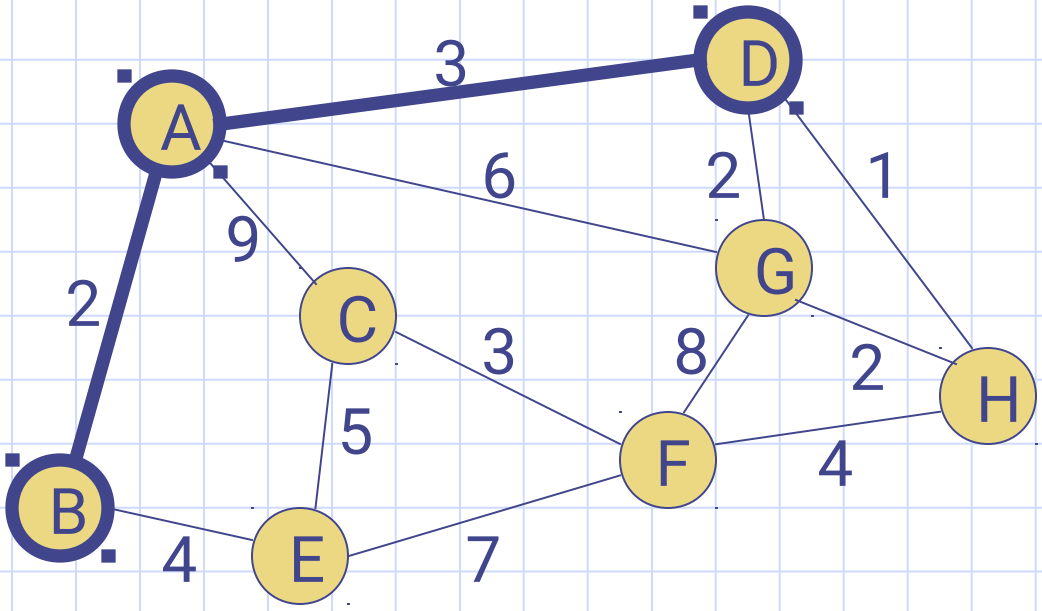
Algoritmul lui Prim

- U = {A}, N\U = {B, C, D, E, F, G, H}
- Arcele care leaga noduri din multimea U cu noduri din multimea N\U sunt: A-B, A-C, A-D, A-E, A-F, A-G, A-H. Cel mai ieftin dintre ele este A-B deci B este urmatorul nod ales (daca vreun arc dintre cele enumerate nu exista fizic in graf, costul lui se considera infinit)



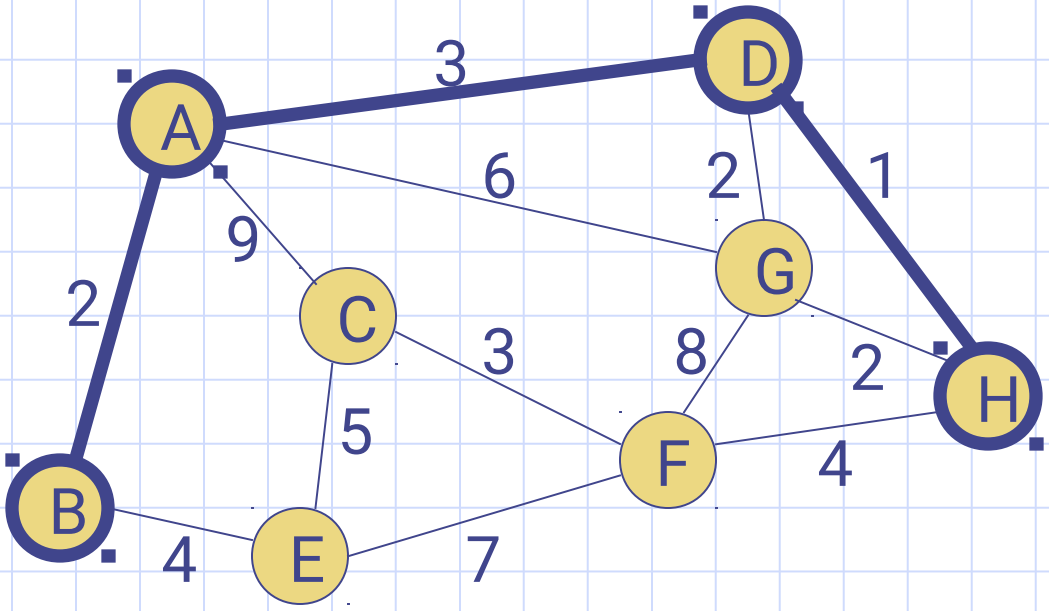
Algoritmul lui Prim

- U = {A, B}, N\U = {C, D, E, F, G, H}
- Arcele care leaga noduri din multimea U cu noduri din multimea N\U sunt: A-C, A-D, A-E, A-F, A-G, A-H, B-C, B-D, B-E, B-F, B-G, B-H. Cel mai ieftin dintre ele este A-D, deci D este urmatorul nod ales



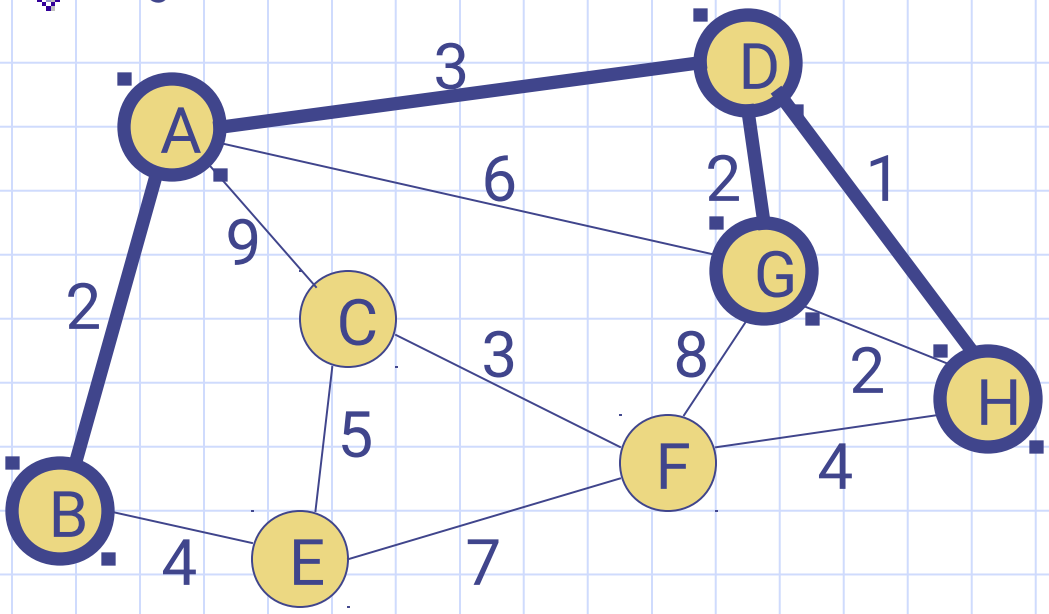
Algoritmul lui Prim

- U = {A, B, D}, N\U = {C, E, F, G, H}
- Se alege nodul H si arcul D-H



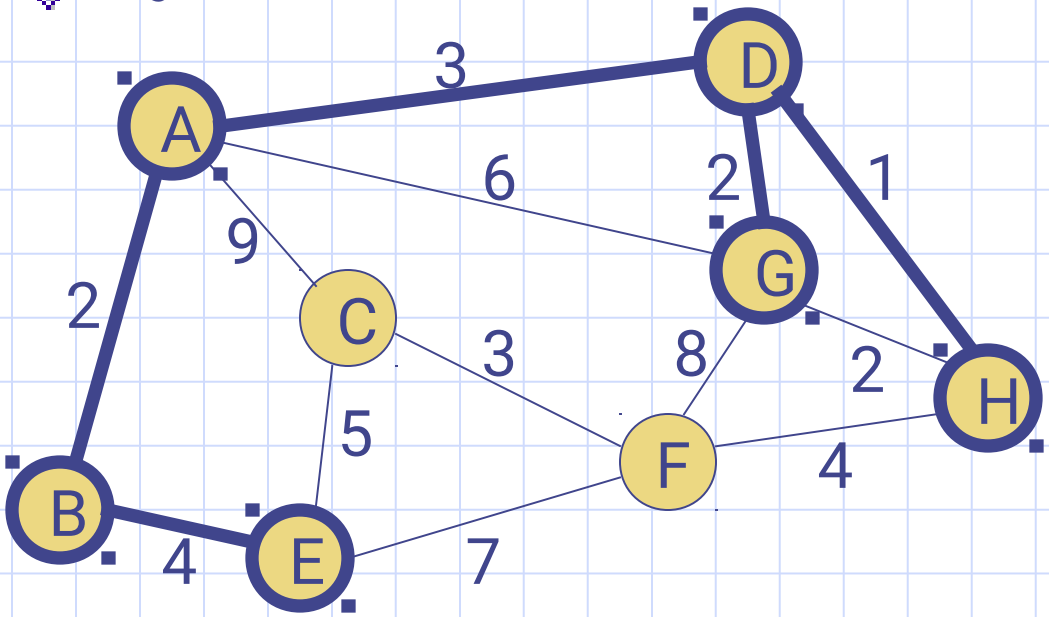
Algoritmul lui Prim

- U = {A, B, D, H}, N\U = {C, E, F, G}
- Sunt doua posibilitati, ambele de cost 2:
  - Nodul G si arcul D-G
  - Nodul G si arcul G-H
- Alegerea este arbitrara, vom continua cu nodul G si arcul D-G



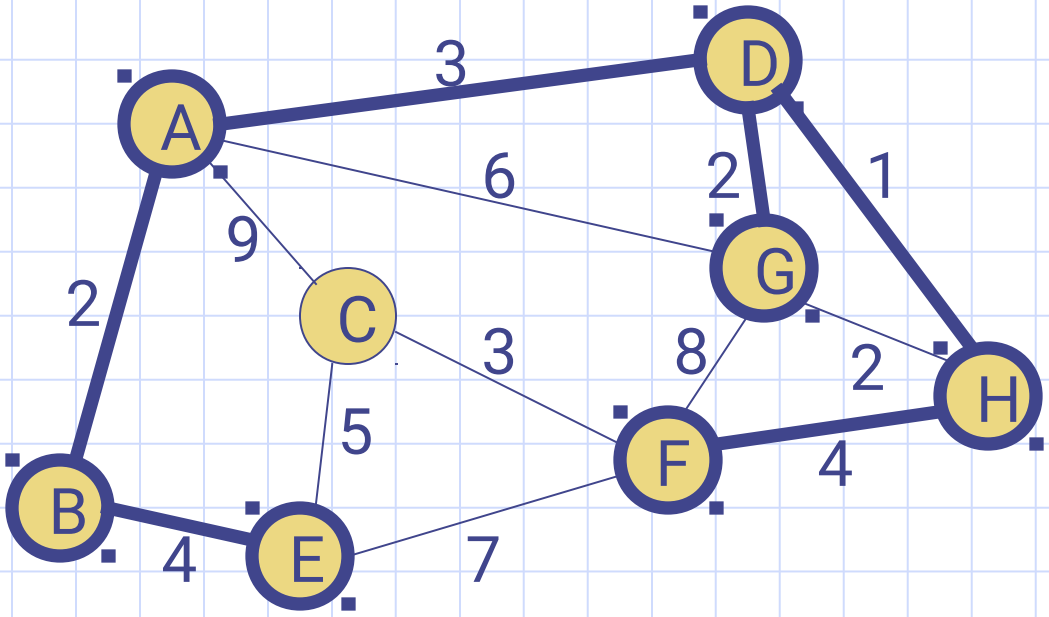
Algoritmul lui Prim

- U = {A, B, D, G, H}, N\U = {C, E, F}
- Sunt doua posibilitati, ambele de cost 4:
  - nodul E si arcul B-E
  - nodul F si arcul F-H
- Alegerea este arbitrara, vom continua cu nodul E si arcul B-E



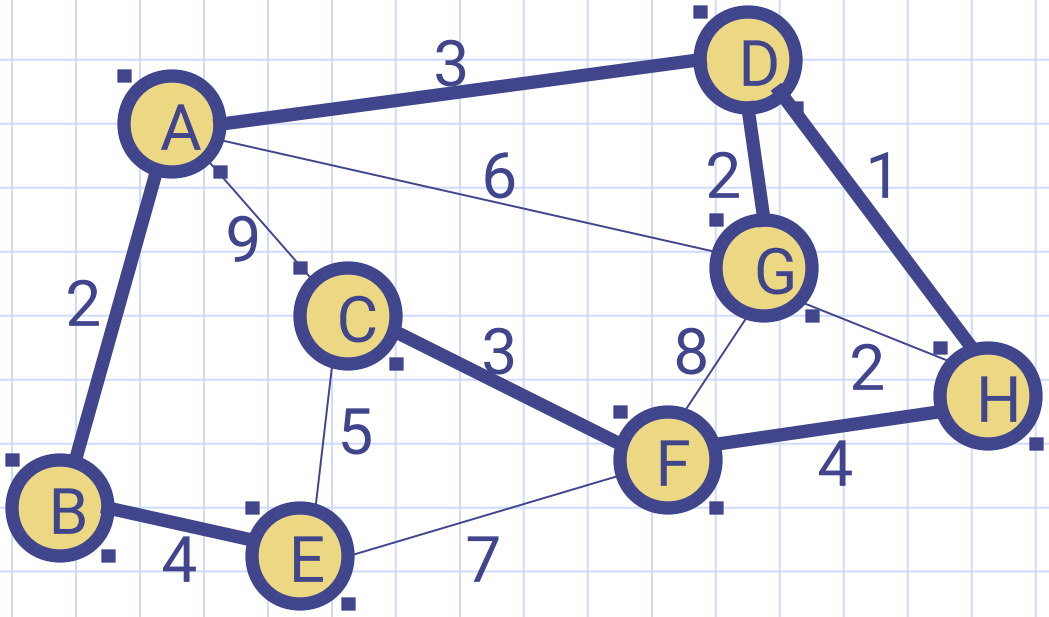
Algoritmul lui Prim

- U = {A, B, D, E, G, H}, N\U = {C, F}
- Se alege nodul F si arcul F-H



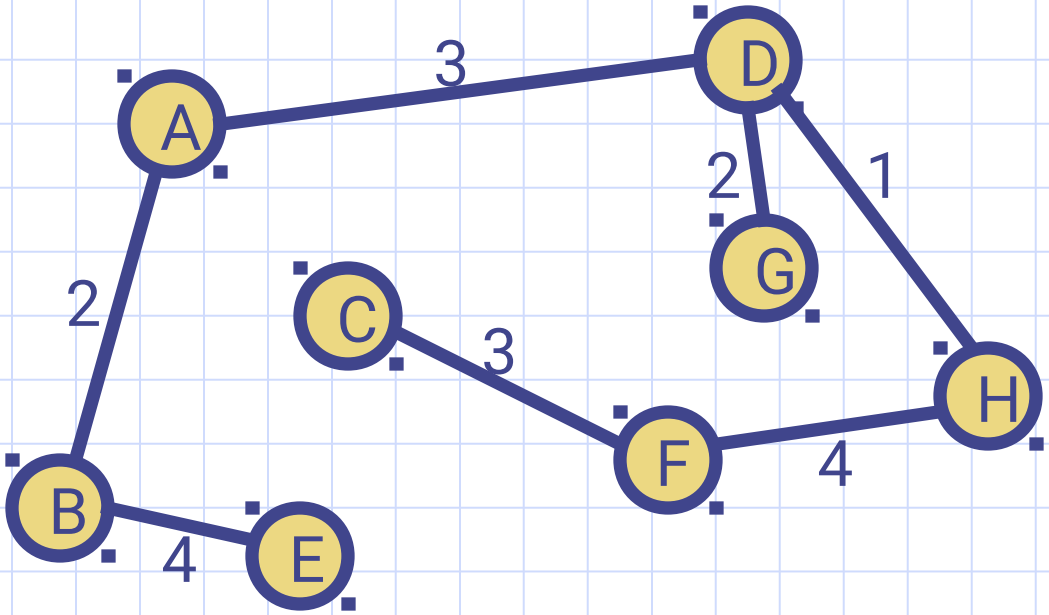
Algoritmul lui Prim

- U = {A, B, D, E, F, G, H}, N\U = {C}
- Se alege nodul C si arcul C-F



Algoritmul lui Prim

- Algoritmul se incheie deoarece U = N
- Arborele de acoperire este cel din figura de mai jos, costul sau fiind 19
- Nu exista un arbore de acoperire mai ieftin pentru graful dat



Algoritmul lui Prim

- Arborele obtinut este generalizat, dar denaturat, deoarece arcele sale se prezinta chiar in pozitiile in care existau ele in graful initial
- Daca se doreste, nodurile si arcele alese pot fi aduse la forma de arbore printr-o simpla redesenare, luand oricare nod ca si radacina si vecinii sai pe post de fii
- Pe slide-ul urmator, arborele de acoperire gasit este redesenat intr-o forma mai familiara, de arbore generalizat, considerand nodul A ca radacina si apoi considerand nodul D ca radacina

Algoritmul lui Prim