Search and Pathfinding Algorithms
Algoritmii de căutare in grafuri

- Algoritmii de căutare in graf exploreaza un graf pentru descoperire generală, fie pentru căutare explicită.
- Acești algoritmi parcurg căi prin graf dar nu există nicio așteptare ca aceste căi sa fie optime din punct de vedere computațional.
- Breadth și Depth Search, cătarea în lățime și adâncime sunt fundamentale pentru traversarea unui graf și sunt adesea un prim pas necesar pentru multe alte tipuri de analize.
- Algoritmii de găsire a căilor se bazeaza pe algoritmii de cautare în graf și exploreaza rutele dintre noduri, începând de la un nod și traversând relațiile pâna la destinație.
- Acești algoritmi sunt utilizați pentru a **identifica rutele optime** din graf în planificarea logistica, rutarea apelurilor sau IP cu cel mai mic cost și simularea jocurilor.

Algoritmii de pathfinding prezentați:

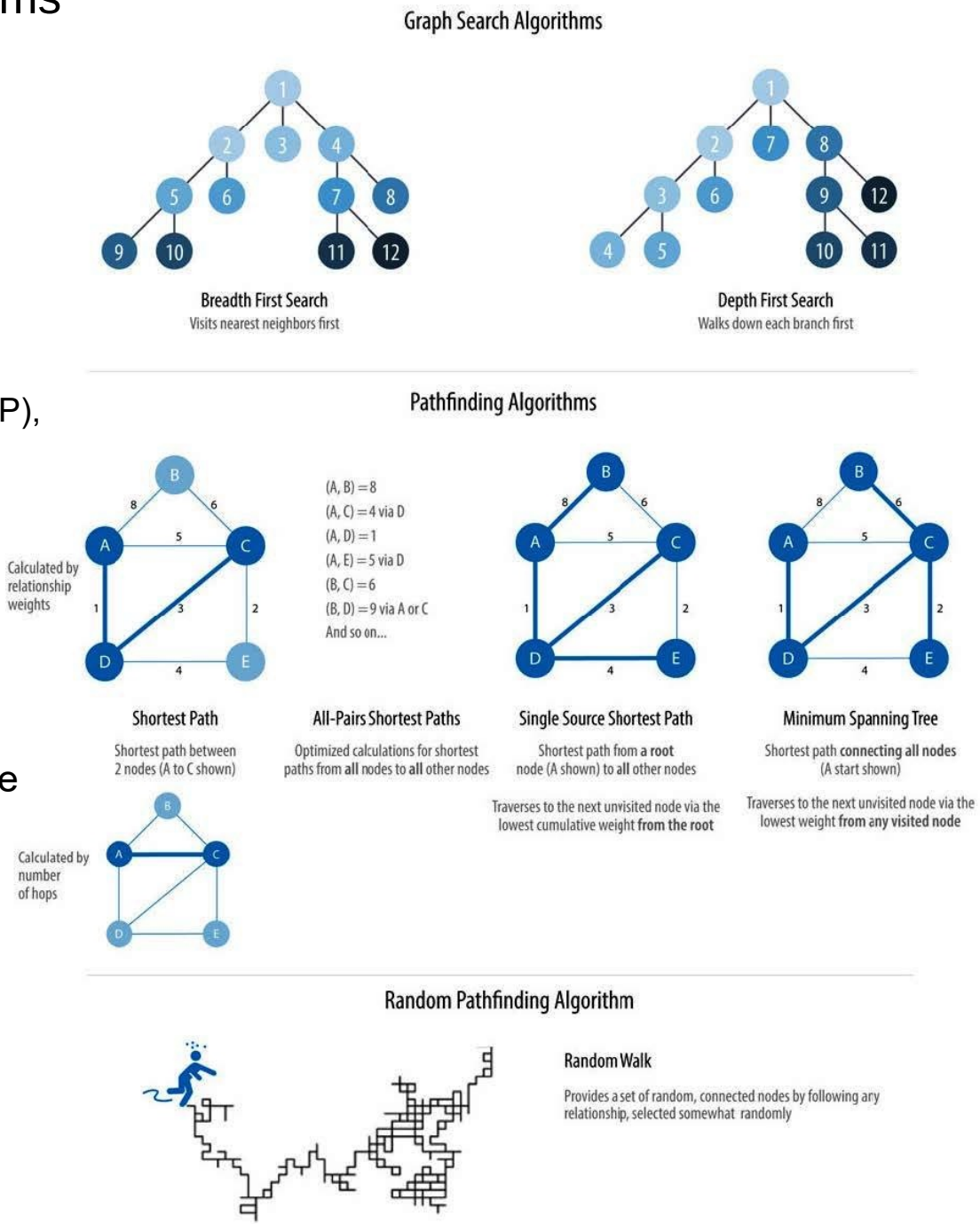
- Shortest Path, cel mai scurt drum cu doua variante utile (A* și Yen): găsirea celei mai scurte căi sau a celor mai scurte căi între doua noduri alese
- All Pairs Shortest Path and Single Source Shortest Path, pentru gasirea celor mai scurte cai între toate perechile sau de la un nod ales la toate celelalte
- Minimum Spanning Tree, Arborele minim de acoperire: pentru gasirea unei structuri arborescente conectate cu cel mai mic cost pentru vizitarea tuturor nodurilor dintr-un nod ales
- Random Walk: deoarece este un pas util de preprocesare / eșantionare pentru fluxurile de lucru de învățare automată a algoritmilor grafurilor

Search and Pathfinding Algorithms

- Figura următoare prezinta diferențele dintre aceste tipuri de algoritmi
  - **Graph search** algorithms BFS, DFS
  - **Patfinding algorithms** Shorest Path (SP), All Pairs Shortest path (APSP), Single Source Shortest path (SSSP), Minimum Spanning Tree (MST), RandomWalk

- Pentru fiecare algoritm, se începe cu o scurta descriere a algoritmului și orice informații pertinente despre modul în care funcționeaza.

- Sunt incluse îndrumări cu privire la momentul utilizarii algoritmilor asociați.



Pathfinding and graph search algorithms
Algoritmi de Căutare și Găsire Drumuri in Grafuri

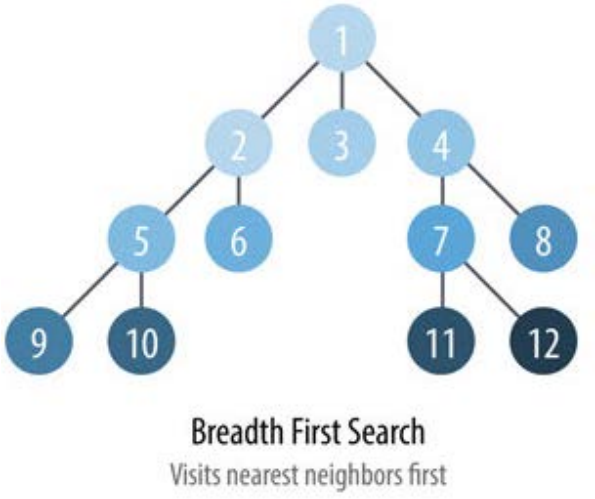
Tabel – sumar al algoritmilor cu un exemplu de utilizare.

Algoritm	Ce realizează	Exemple	App Neo4J
Breadth First Search Cătare în lățime	Traverseaza o structura de arbore (tree) cautand (nearest neighbors) cei mai apropiati vecini si vecinii de la nivel inferior.	Localizare noduri vecine in sisteme GPS pentru identificarea celor mai apropiate noduri de interes.	Nu
Depth First Search Căutarea în Adâncime	Traversează o strcutura arborescentă (tree) și explorează cât mai adânc posibil fiecare ramură a arborelui (până la frunze dacă este posibil) înainte de backtracking	Descoperă o soluție optimă în simulări de jocuri cu alegeri ierarhice (în ierarhie arborecentă)	Nu
Shortest Path (SP) Cel mai scurt drum (cale) Variații: A*, Yen	Calculează cel mai scurt drum între o pereche de noduri	Găsirea direcțiilor de deplasare (ex. automobil) între două locații	Da
All Pairs Shortest Path (APSP) Toate Drumurile (căile) cele mai scurte	Calculează cele mai scurte drumuri (căi) între toate perechile de noduri din graf	Evaluează rute alternative într-un trafic aglomerat.	Da
Single Source Shortest Path (SSSP) Cel mai scurt drum de la un singur nod sursă (rădăcină)	Calculează cel mai scurt drum între un singur nod (rădăcină) și toate celelalte noduri	Cel mai mic cost al rutării apelurilor în rețele de telefonie	Da
Minimum Spanning Tree (MST) Arbore de acoperire minimă	Calculează drumul într-un arbore (tree) cu structură conectată cu cel mai mic cost de vizitare al tuturor nodurilor.	Optimizarea rutelor conectate ex. întindere cabluri telefonie, strangere deșeuri în oraș	Da
Random Walk Deplasare aleatorie	Returnează lista de noduri și drumul de o distanță specificată prin alegerea aleatorie a relațiilor de traersat.	Augmentarea învățării pentru machine learning (ML) sau date pentru algoritmica grafurilor	Da

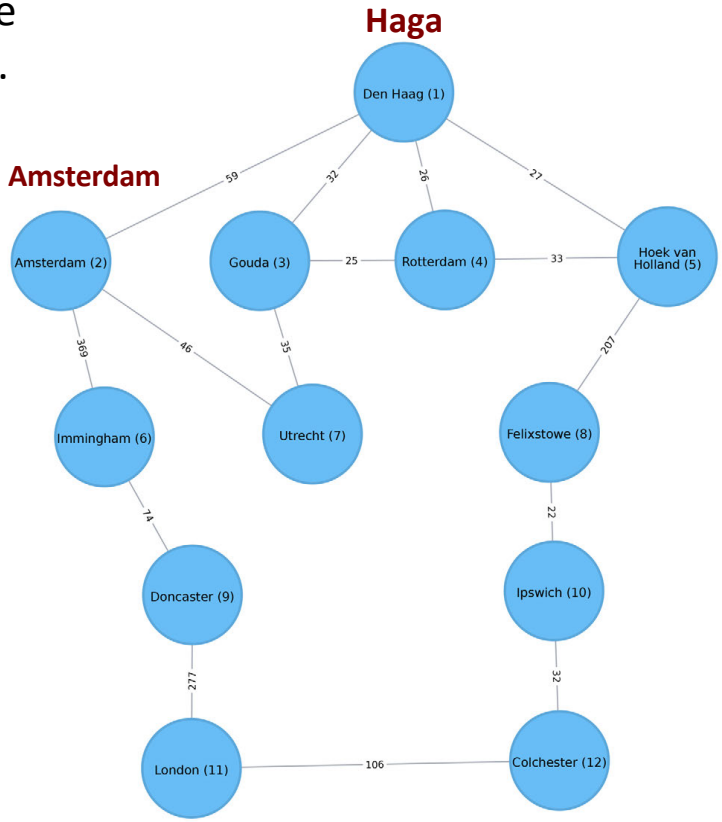
Breadth First Search (BFS)

Parcurgerea în Lățime a arborilor (trees)

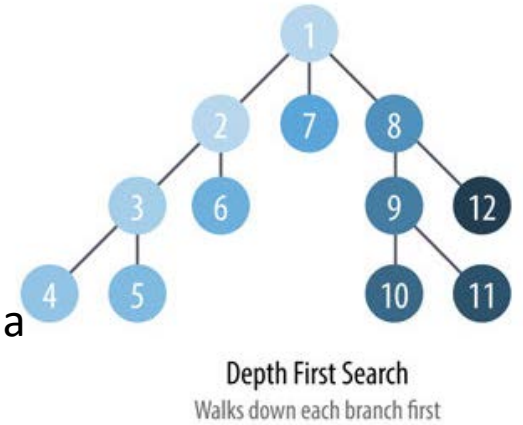
- Breadth First Search (BFS) este un algoritm fundamentali de traversare a grafurilor. Se pornește de la un nod ales și explorează toți vecinii săi la un salt (hop) distanță înainte de a vizita toți vecinii la doua salturi (hopuri) distanță și continuă similar mai departe
- A fost publicat in 1959 de E.F. Moore și l-a folosit pentru a gasi cea mai scurta cale de ieșire dintr-un labirint.
- A fost dezvoltat într-un algoritm de rutare a firelor de C. Y. Lee 1961, “An Algorithm for Path Connections and Its Applications”.



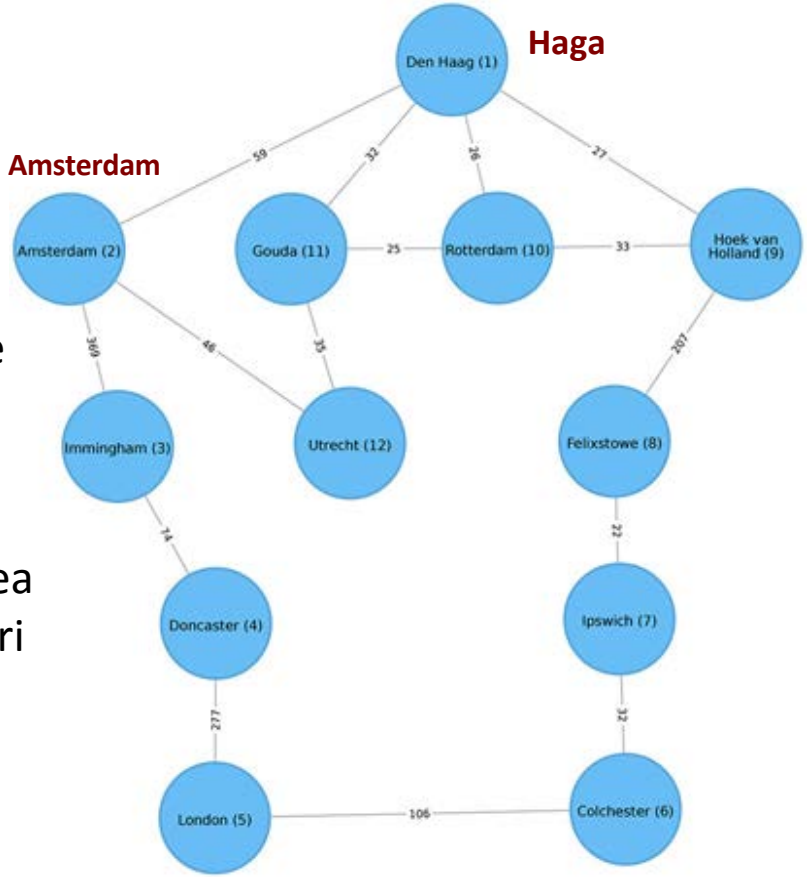
- BFS este cel mai utilizat ca bază pentru alți algoritmi orientați spre obiective, ex. **Shortest Path, Connected Components și Closeness Centrality folosesc algoritmul BFS.**
- Poate fi folosit și pentru a gasi cea mai scurta cale între noduri.
- Figura arată ordinea în care am vizita nodurile grafului de transport daca se caută amplu din orașul de start olandez, Den Haag (Haga).
- Numerele de lângă numele orașului indică ordinea în care este vizitat fiecare nod.
- Mai întâi vizitam toți vecinii direcți ai Den Haag, înainte de a-i vizita pe vecinii lor și pe vecinii vecinilor lor, pâna când nu mai sunt relații de traversat.



Depth First Search (DFS)
Parcurgere arbore în adâncime



- Depth First Search (DFS) este un algoritm fundamental de traversare a grafului.
- Începe de la un nod ales, alege unul din vecinii săi și apoi traversează cât de departe posibil de-a lungul acelei căi înainte de a se întoarce.
- DFS a fost inventat de matematicianul francez Charles Pierre Trémaux ca strategie pentru rezolvarea labirinturilor. Acesta este util pentru a simula posibile căi de modelarea scenariilor.
- Figura arată ordinea în care se vizitează nodurile grafului de transport daca DFS a pornit din Haga.
- Se observă cât de diferita este ordinea nodurilor comparativ cu BFS.
- Pentru acest DFS, începem prin traversarea de la Haga (**radacina** grafului) la Amsterdam și apoi se poate ajunge la orice alt nod din graf fara sa ne întoarcem.
- Se observă cum algoritmii de căutare pun bazele deplasării prin grafuri.
- Să observăm algoritmii de găsire a căilor ce găsesc cea mai ieftină cale în ceea ce privește numărul de noduri sau greutatea.
- Greutățile pot fi orice măsură (ex. timpul, distanța, capacitatea sau costul.)

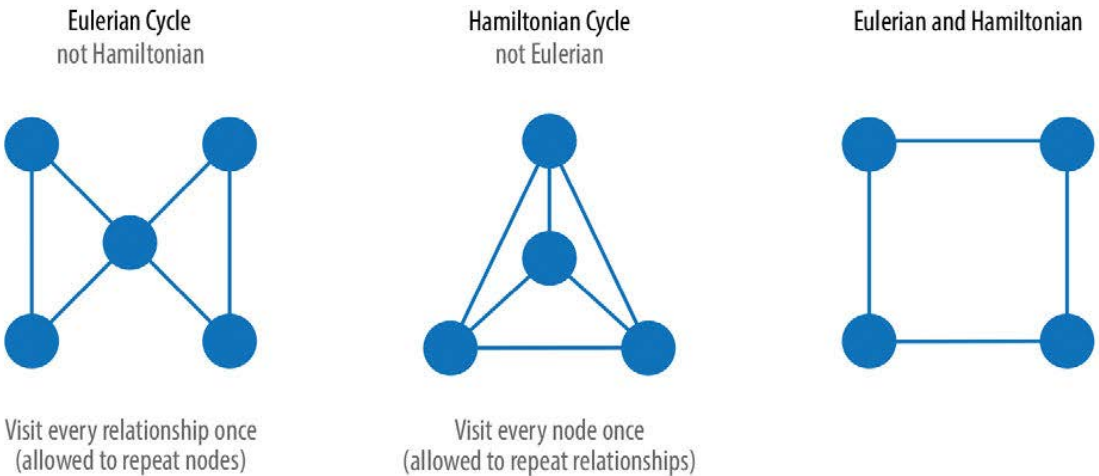


Two Special Paths/Cycles

Doua trasee/cicluri speciale

Doua trasee/cicluri speciale există în analiza grafurilor:
1) drum **eulerian** ce este una în care fiecare relație este vizitata exact o dată.
2) drum **Hamiltonian** ce este una în care fiecare nod este vizitat exact o data.
Un drum poate fi atât eulerian, cât și hamiltonian, iar daca începeți și terminați la același nod, este considerat **un tur**.
Comparație vizuală este prezentata în Figura.

- Problema podurilor Königsberg caută un ciclu eulerian. Este ușor de văzut cum se aplica acest lucru scenariilor de rutare, cum ar fi direcționarea plugurilor de zapada și livrarea corespondenței.



- Drumurile euleriene sunt utilizate și de alți algoritmi în procesarea datelor în arbori și sunt simple din punct de vedere matematic de studiat decât altele.

- **Ciclul Hamiltonian** este cunoscut cu problema comis voiajorului (TSP): "Care este cea mai scurtă rută posibilă pentru un agent de vânzari sa viziteze fiecare dintre orașele atribuite și să se întoarcă în orașul de origine?"

- Deși aparent similar cu un **tur eulerian,** TSP este dificil computațional cu alternative de aproximare.
- Este utilizat într-o mare varietate de probleme de planificare, logistică și optimizare.

Shortest Path (SP) Drumul cel mai scurt

- Alg. Shortest Path **calculează cea mai scurta cale (ponderată) dintre o pereche de noduri.**
- **Utilizat** pentru interacțiunile utilizatorilor și fluxurile de lucru dinamice, deoarece funcționeaza în timp real.
- Pathfinding datează din secolul al 19-lea.Este considerata problema clasica de grafuri.
- A câștigat importanță la începutul anilor 1950 în contextul rutelor alternative; adică găsirea celei de-a doua rute cele mai scurte dacă cea mai scurta ruta este blocata.

În 1956, Edsger Dijkstra a creat cel mai cunoscut dintre acești algoritmi.

- **Algoritmul Dijkstra Shortest Path** funcționeaza găsind mai întâi relația cu cea mai mică greutate de la nodul de pornire la nodurile conectate direct.
- Se ține evidența acestor greutați și se deplaseaza la nodul "cel mai apropiat".
- Apoi efectueaza același calcul, dar acum ca total cumulativ din nodul de pornire.
- Algoritmul continuă similar evaluând un "val" de greutăți cumulate și alegând întotdeauna cea mai mica cale cumulativă ponderată pentru a avansa, pâna când ajunge la nodul destinație.

Se observă în analiza grafurilor utilizarea termenilor greutate, cost, distanța și hops atunci când descrieți relații și căi.

- Ponderea/ȚGreutatea" este valoarea numerica a unei anumite proprietați a unei relații.
- "Cost" este folosit în mod similar atunci când luam în considerare greutatea totală a unei căi.
- "Distanța" se utilizează ca nume al proprietații relației ce indică costul traversării între o pereche de noduri. Nu este necesar să fie măsură fizica reala a distanței.
- Hops/salturi exprimă numărul de relații dintre două noduri. Sunt și termeni combinați, ex. "Este o distanță de cinci hops pâna la Londra" sau "Acesta este cel mai mic cost al distanței"

Când se utilizează Shorest Path (SP), drumul cel mai scurt în graf ?

- U tilizați SP pentru a gasi rute optime între o pereche de noduri, pe baza numarului de salturi/hops sau a oricarei valori ponderate (greutate/weight) a relației.
- De exemplu, poate oferi raspunsuri în timp real despre gradele de separare, cea mai scurta distanță dintre puncte sau ruta cea mai puțin costisitoare.
- Utilizați SP pentru a explora pur și simplu conexiunile dintre anumite noduri.

Exemple de cazuri de utilizare:

- Gasirea indicațiilor de orientare între locații. Instrumentele de cartografiere web, cum ar fi Google Maps, utilizează algoritmul Shortest Path sau o varianta apropiata pentru a oferi indicații rutiere.
- Găsirea gradelor de separare între oameni în rețelele sociale. De exemplu, atunci când vizualizați profilul cuiva pe LinkedIn, acesta va indica câte persoane va separa în graf, precum și listează conexiunile reciproce.
- Găsirea numarului de grade de separare dintre un actor și Kevin Bacon pe baza filmelor în care au aparut (numarul Bacon). Un exemplu în acest sens poate fi vazut pe site-ul Oracle of Bacon. Proiectul Erdös Number ofera o analiza a grafului similară bazata pe colaborarea cu Paul Erdös, unul dintre cei mai prolifici matematicieni ai secolului al XX-lea.

Observație Algoritmul Dijkstra **nu suporta ponderi negative.**

- Algoritmul presupune ca adaugarea unei relații la o cale nu poate face niciodata o cale mai scurtă - un invariant care ar fi încalcat cu ponderi negative.

Shortest Path Variation: A* (variantă Shortest Path)

- Algoritmul A* Shortest Path îmbunatățește algoritmul lui Dijkstra prin găsirea mai rapida a celor mai scurte cai.
- Se realizează prin includerea de informații suplimentare pe care algoritmul le poate utiliza, ca parte a unei funcții euristice, la determinarea căilor de explorat continuare.
- Algoritmul a fost inventat de P.Hart, N. Nilsson și B.Raphael și descris în 1968 (“A Formal Basis for the Heuristic Determination of Minimum Cost Paths”)
- Algoritmul A* determină care din căile parțiale să se extindă la fiecare iterație a buclei sale principale.
- Se folosește o estimare a costului (euristic) rămas pentru a ajunge la nodul țintă.

- Observație- Atenție la euristica folosită pentru a estima costurile căii.
- Subestimarea costurilor traseului poate include în mod inutil unele căi ce ar fi putut fi eliminate, dar rezultatele vor fi totuși exacte.
- Cu toate acestea, dacă euristica supraestimează costurile, poate sări peste căile reale mai scurte (estimate incorect a fi mai lungi) care ar fi trebuit să fie evaluate, ceea ce poate duce la rezultate inexacte.

A* selecteaza calea ce minimizează urmatoarea funcție: f(n)=g(n)+h(n)

g(n) = costul caii de la nod de start la nodul n.
h(n) = costul estimat al căi de la nodul n la nodul destinație, calculat prin euristica.

- În unele implementari **distanța geospațială** este utilizata ca euristică.
- În exemplul cu set de date de transport, se utilizează **latitudinea și longitudinea** fiecarei locații ca parte a funcției euristice.

Shortest Path Variation **Yen's k-Shortest Paths** (variantă Shortest Path)

- Algoritmul k-Shortest Paths al lui Yen este similar cu algoritmul Shortest Path, dar găsește nu doar cea mai scurtă cale între doua perechi de noduri, ci**, în plus calculează și a 2-a cea mai scurtă cale, a 3-a cea mai scurta cale și așa mai departe pâna la k-1 abateri ale celor mai scurte căi.**

- J. Y. Yen a inventat algoritmul în 1971 ("Finding the K Shortest Loopless Paths in a Network".)
- Algoritmul este util pentru a obține căi alternative atunci când găsirea celei mai scurte căi absolute nu este singurul obiectiv.
- Este util atunci când este necesar mai mult de un plan de rezervă.

All Pairs Shortest Path (APSP)

- Algoritmul All Pairs Shortest Path (APSP) calculeaza cea mai scurtă cale (ponderata) dintre toate perechile de noduri.
- Este mai eficient decât rularea algoritmului Single Source Shortest Path pentru fiecare pereche de noduri din graf.
- APSP optimizeaza operațiunile ținând evidența distanțelor calculate pâna acum și rulând pe noduri în **paralel.**
- Aceste distanțe cunoscute pot fi apoi reutilizate atunci când se calculează cea mai scurtă cale catre un nod nevazut.
- Se poate utiliza exemplul următor pentru a înțelege funcționarea algoritmului.
- Este posibil ca unele perechi de noduri să nu fie accesibile unul pentru celălalt, adică nu există cea mai scurtă cale între aceste noduri.
- Algoritmul nu returnează distanțe pentru aceste perechi de noduri.

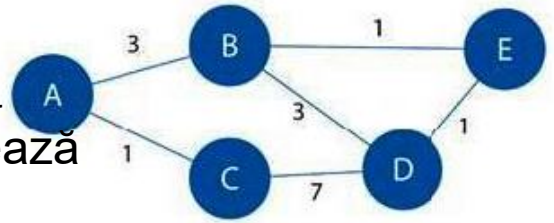
All Pairs Shortest Path (APSP)

Detalii de calcul

APSP este mai ușor de înțeles când se urmează o secvența de operații. Diagrama Figura parcurge pașii pentru nodul A.

A. Inițial, algoritmul presupune o distanța infinita față de toate nodurile. Când este selectat un nod de pornire, atunci distanța pâna la acel nod este setata la 0. Calculul continuă:

1. De la nodul de start A evaluam costul trecerii la nodurile la care se poate ajunge și se actualizează aceste valori.



Pentru cea mai mica valoare, se poate alege între B (cost 3) sau C (cost 1). C este selectat pentru urmatoarea traversare.

2. Din nod C, alg. actualizează distanțele cumulate de la A la noduri la care se poate ajunge direct din C.

Valorile sunt actualizate numai atunci când a fost găsit un cost mai mic: A=0, B=3, C=1, D=8, E=8

All nodes start with a ∞ distance and then the start node is set to a 0 distance			Each Step Keeps or Updates to the Lowest Value Calculated so Far Only steps for node A to all nodes shown				
			1st from A	2nd from A to C to Next	3rd from A to B to Next	4th from A to E to Next	5th from A to D to Next
A	∞	0	0	0	0	0	0
B	∞	∞	3	3	3	3	3
C	∞	∞	1	1	1	1	1
D	∞	∞	∞	8	6	5	5
E	∞	∞	∞	∞	4	4	4

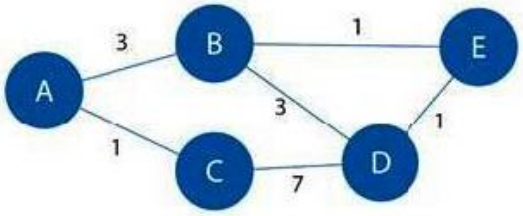
Etapele de calcul APSP din nodul de start A spre toate celelate noduri cu updates colorate.

All Pairs Shortest Path (APSP)

3. B este selectat ca urmatorul nod cel mai apropiat nevizitat. B are relații cu nodurile A, D și E. Alg. calculează distanța pâna la acele noduri prin însumarea distanței de la A la B cu distanța de la B la fiecare dintre aceste noduri. (Cel mai mic cost de la nodul de pornire A la nodul curent este întotdeauna păstrat ca și cost irecuperabil.) Rezultate calcul distanța (d):

d(A,A) = d(A,B) + d(B,A) = 3 + 3 = 6
d(A,D) = d(A,B) + d(B,D) = 3 + 3 = 6
d(A,E) = d(A,B) + d(B,E) = 3 + 1 = 4

- Aici, distanța de la nodul A la B și înapoi la A, afișata ca d(A,A) = 6, este mai mare decât cea mai scurta distanță deja calculata (0), deci valoarea sa nu este actualizata.
- Distanțele pentru nodurile D (6) și E (4) sunt mai mici decât distanțele calculate anterior, a.i. valorile lor sunt actualizate.



4. Nodul E este selectat în continuare. Doar totalul cumulat pentru ajungerea la D (5) este acum mai mic și astfel este singurul actualizat.

5. Când D este evaluat în final, nu există noi greutăți minime ale căii; nimic nu se mai actualizează, iar algoritmul se termină.

			Each Step Keeps or Updates to the Lowest Value Calculated so Far Only steps for node A to all nodes shown				
All nodes start with a ∞ distance and then the start node is set to a 0 distance			1st from A	2nd from A to C to Next	3rd from A to B to Next	4th from A to E to Next	5th from A to D to Next
A	∞	0	0	0	0	0	0
B	∞	∞	3	3	3	3	3
C	∞	∞	1	1	1	1	1
D	∞	∞	∞	8	6	5	5
E	∞	∞	∞	∞	4	4	4

Etapele de calcul APSP din nodul de start A spre toate celelate noduri cu updates colorate.

All Pairs Shortest Path (APSP)

Observație - Chiar daca algoritmul All Pairs Shortest Path este optimizat pentru a rula calcule în paralel pentru fiecare nod, se poate extinde pentru un graf foarte mare.
Se ia în considerare utilizarea unui subgraf daca trebuie doar să evaluați căile dintre o subcategorie de noduri.

Când se utilizează ASPS?

- ASPS se utilizează pentru înțelegerea rutelor alternative atunci când ruta cea mai scurtă este blocată sau devine suboptimală.
- De exemplu, acest algoritm este utilizat în planificarea logică a rutelor pentru a asigura cele mai bune căi multiple pentru rutarea diversității.
- Utilizați ASPS atunci când trebuie sa luați în considerare toate rutele posibile între toate sau majoritatea nodurilor dvs.

Exemple de cazuri de utilizare includ:

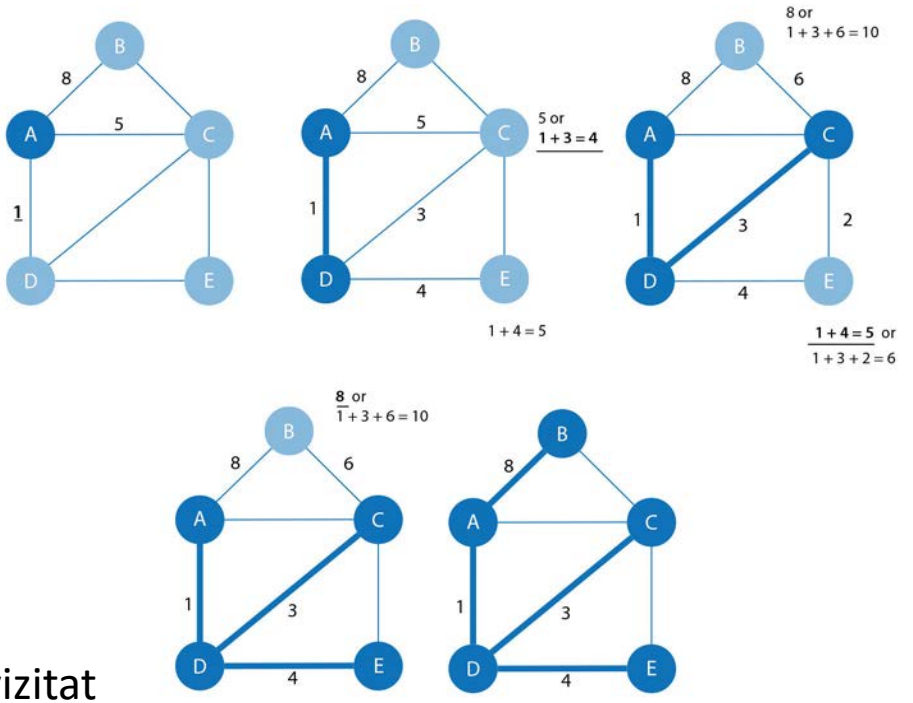
- Optimizarea amplasarii facilitaților urbane și a distribuției de bunuri.

Exemplu - determinarea încărcăturii de trafic așteptate pe diferite segmente ale unei rețele de transport (cartea R. C. Larson și A. R. Odoni, Urban Operations Research, Prentice-Hall)

- Găsirea unei rețele cu lățime de bandă maximă și latență minimă ca parte a unui algoritm de proiectare a centrului de date. Exista mai multe detalii despre aceasta abordare în lucrarea "REWIRE: An Optimization-Based Framework for Data Center Network Design”, by A. R. Curtis et al.

Single Source Shortest Path (SSSP)

- Algoritmul Single Source Shortest Path (SSSP), a apărut aproximativ în același timp cu algoritmul Shortest Path al lui Dijkstra, acționeaza ca o implementare pentru ambele probleme.
- Algoritmul SSSP calculează cea mai scurtă cale (ponderată) de la un nod rădăcină la toate celelalte noduri din graf, așa cum se demonstrează în Figura cu Etapele SSSP.



Se procedeaza astfel::

1. Se pornește cu un nod radacina din care vor fi masurate toate caile. În Figura s-a selectat nodul A ca radacina.
2. Relația cu cea mai mica greutate provenita de la acel nod radacina este selectata și adaugată arborelui, împreuna cu nodul sau conectat, aici d(A,D)=1.
3. Urmatoarea relație cu cea mai mica greutate cumulata de la nodul radacina la orice nod nevizitat este selectata și adaugata arborelui în același mod.

Figura d(A,B)=8, d(A,C)=5 direct sau 4 via A-D-C, și d(A,E)=5. Deci, traseul prin A-D-C este ales și C este adaugat la arbore.

4. Procesul continuă pâna când nu mai exista noduri de adaugat și există cea mai scurta cale a sursei unice, SSSP.

Single Source Shortest Path (SSSP)

Când se utlizează SSSP?

- Utilizați SSSP la evaluare ruta optimă de la un punct de pornire fix la toate celelalte noduri individuale. Deoarece ruta este aleasa pe baza greutații totale (costului total) a drumului de la radacina, este utila pentru gasirea celei mai bune cai catre fiecare nod, dar nu neaparat atunci când toate nodurile trebuie vizitate într-o singura parcurgere.

- De exemplu, SSSP este util pentru identificarea principalelor rute de utilizat pentru **serviciile de urgența** în cazul în care nu vizitați fiecare locație pentru fiecare incident, dar nu pentru a gasi o singura ruta pentru colectarea gunoiului unde trebuie sa vizitați fiecare casă într-un singur drum (În ultimul caz, veți utiliza algoritmul Minimum Spanning Tree.)

Exemple de cazuri de utilizare includ:

- Detectarea modificarilor în topologie, cum ar fi erorile legaturilor, și sugerarea unei noi structuri de rutare în câteva secunde

- Utilizarea Dijkstra ca protocol de rutare IP pentru utilizare în sisteme autonome, cum ar fi o rețea locala (LAN)

Minimum Spanning Tree (MST)

- Minimum Spanning Tree algorithm, algoritmul pentru determinarea arborelui de intindere (greutate) minima pornește de la un nod dat și gasește toate nodurile sale accesibile și setul de relații ce conecteaza nodurile împreuna cu greutatea minima posibilă.
- Traverseaza de la orice nod vizitat spre urmatorul nod nevizitat cu cea mai mică greutate, evitând ciclurile.
- Primul algoritm cunoscut Minimum Weight Spanning Tree a fost dezvoltat de cercetatorul ceh Otakar Boruvka în 1926.
- Algoritmul lui Prim, inventat în 1957, este un alg. de bază și cel mai cunoscut.
- **Algoritmul lui Prim's** este **similar cu algoritmul Shortest Path al lui Dijkstra**, dar în loc sa minimizeze lungimea totala a unei căi care se termină la fiecare relație, minimizeaza lungimea fiecarei relații individual.
- MST tolereaza relațiile cu **greutate negativa** **(**Alg**.** Djiktra nu tolerează greutăți negative)

Minimum Spanning Tree (MST)

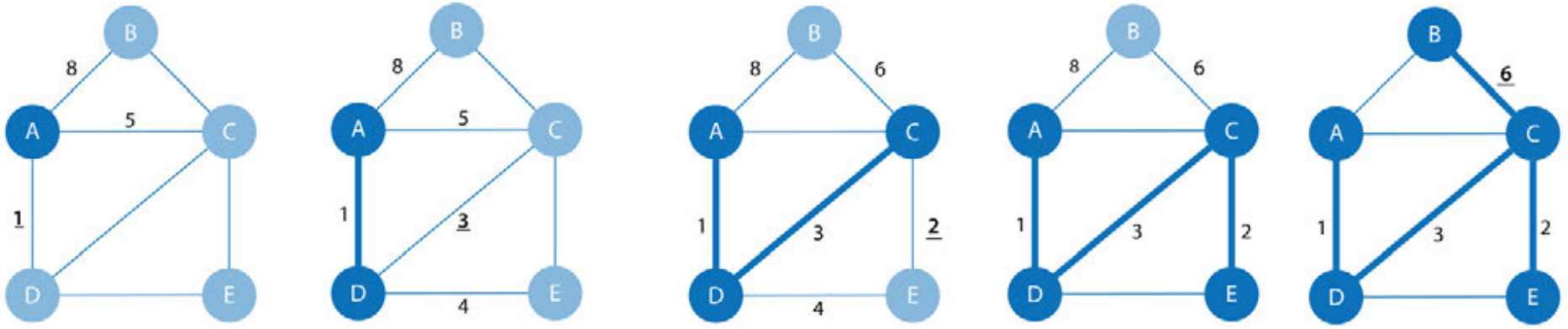
- Etapele Minimum Spanning Tree algorithm ilustrate în Figura

Etapele MST

1. Începe cu un arbore care con?ine un singur nod. În Figura începem cu nodul A.
2. Relația cu cea mai mica greutate provenind de la acel nod este selectata și adaugata în arbore (împreuna cu nodul sau conectat). În acest caz, A-D.
3. Acest proces se repeta, alegând întotdeauna relația minima-greutate care unește orice nod care nu este deja în arbore.

Daca comparați cu exemplul SSSP se observa ca în al 4-lea graf căile devin diferite - deoarece SSSP evalueaza calea cea mai scurta pe baza totalurilor cumulate de la radacina, în timp ce MST analizeaza doar costul urmatorului pas.

4. Când nu mai exista noduri de adaugat, arborele este de întindere minima, MST.



Minimum Spanning Tree (MST)

Observație MST are rezultate semnificative când este rulat pe un graf în care relațiile au ponderi diferite. **Daca graful nu are ponderi sau toate relațiile au aceeași pondere, atunci orice arbore de întindere este un arbore de întindere minim.**

Exista, variante ale acestui algoritm ce gasesc arbore de întindere cu greutate maxima (arborele cu cel mai mare cost) și arbore de întindere k (dimensiune arbore limitata).

Când se utilizează arborele de întindere minima MST?

- Utilizați MST pentru aflarea celei mai bune rute vizitând toate nodurile. Deoarece traseul este ales în funcție de costul fiecarui pas urmator, este util atunci când trebuie sa vizitați toate nodurile într-un singur drum.

- Alg. se poate utiliza pentru optimizarea căilor pentru sistemele conectate, ex. conducte de apa și proiectarea circuitelor.
- Este util pentru a aproxima unele probleme cu timpi de calcul necunoscuti, ex.Problema comis voiajorului (TSP, Traveling Salesman problem) și tipuri de probleme de rotunjire.

Exemple de cazuri de utilizare includ:

- Minimizarea costurilor de calatorie pentru explorarea unei țari. . “An Application of Minimum Spanning Trees to Travel Planning” descrie modul în care algoritmul a analizat conexiunile aeriene și maritime pentru a face acest lucru.
- Vizualizarea corelațiilor dintre randamentele valutare. Acest lucru este descris în“Minimum Spanning Tree Application in the Currency Market”.
- Urmarirea istoricului transmiterii infecției într-un focar. Pentru mai multe informații, consultați “Use of the Minimum Spanning Tree Model for Molecular Epidemiolog-ical Investigation of a Nosocomial Outbreak of Hepatitis C Virus Infection”.

Random Walk

- Algoritmul Random Walk ofera un set de noduri pe o cale aleatorie într-un graf.
- Termenul a fost menționat pentru prima data de Karl Pearson în 1905 într-o scrisoare adresată revistei Nature, intitulata „The Problem of the Random Walk”.
- Deși conceptul este mai vechi, doar mai recent s-a aplicat Random Walk în Știința rețelelor.
- Random walk, deplasare aleatorie, în general similara unei persoane dezorientate.
- Persoana știe în ce direcție sau în ce punct de final dorește să ajunga, dar pot merge pe o rută foarte întortocheată pentru a ajunge acolo.
- Algoritmul începe de la un nod și urmeaza oarecum aleatoriu una dintre relații înainte sau înapoi catre un nod vecin. Apoi face același lucru din acel nod și așa mai departe, pâna când ajunge la lungimea setata a caii.
- Spunem oarecum aleatoriu, deoarece numarul de relații pe care le are un nod și pe care le au vecinii sai influențeaza probabilitatea prin care un nod va fi parcurs.

Când ar trebui să utilizați Random Walk?

- Utilizați algoritmul Random Walk ca parte a altor algoritmi sau data pipelines atunci când trebuie sa generați un set aleator de noduri conectate.

Exemple de cazuri de utilizare includ:

- Ca parte a algoritmilor node2vec și graph2vec, ce creeaza înglobari de noduri-Aceste înglobari de noduri ar putea fi utilizate ca intrare într-o rețea neuronala.
- Ca parte a detectarii comunității Walktrap, Infomap , daca o plimbare aleatorie returneaza un set mic de noduri în mod repetat, atunci acel set de noduri poate avea o structura comunitara.
- Ca parte a procesului de instruire a modelelor de învațare automata (D. Mack“Review Prediction with Neo4j and TensorFlow”.)
- Alte cazuri de utilizare în N. Masuda et al.“Random Walks and Diffusion on Networks”.

Sumar

- Algoritmii Pathfinding sunt utili pentru înțelegerea modului în care datele sunt
- conectate. S-a început cu algoritmii fundamentali Breadth and Depth First, înainte de a trece la Dijkstra și alți algoritmi pentru găsirea celui mai scurt drum.

- S-au analizat și variante ale algoritmilor de calea cea mai scurta (shortest path algorithms optimized for finding the shortest path from one node to all other nodes or between all pairs of nodes in a graph)

Optimizari pentru a gasi calea cea mai scurta de la un nod la toate celelalte noduri sau între toate perechile de noduri dintr-un graf.

- S-a prezentat și algoritmul Random Walk, ce poate fi folosit pentru a găsi seturi arbitrare de căi.

Resurse

Carte ce acoperă concepte fundamentale și algoritmi ai grafurilor

The Algorithm Design Manual, by Steven S. Skiena(Springer).