IDATT2101 Øving 5

Uvektede grafer

Trang Minh Duong

Deloppgave 1: Bredde-først søk

For å implementere en bredde-først søk implementerte en FIFO-kø med startnode `s`, merk `dist`$[s] = 0$. Når en node u tas ut av køen, besøkes alle naboer v som ikke er oppdaget, disse får `dist`$[v]$ = `dist`$[u] + 1$ og `parent`$[v] = u$, og legges i kø. Siden køen er FIFO, prosesseres noder i ikke-synkende distanse, og første gang vi oppdager v har vi en korteste sti (i antall kanter).

```
1  BFS bfs(const vector<int>& offsets, const vector<int>& to,
       int n, int s) {
2      BFS out{vector<int>(n, -1), vector<int>(n,-1)};
3
4      queue<int> q;
5      out.dist[s] = 0;
6      q.push(s);
7
8      while(!q.empty()) {
9              //Dequeue
10             int u = q.front();
11             q.pop();
12             out.order.push_back(u);
13
14             for(int x : adj[u]) {
15                 if(out.dist[x] == INF) {
16                     out.dist[x] = out.dist[u] + 1;
17                     out.parent[x] = u;
18                     q.push(x);
19             }
20         }
21     }
22     return out;
23 }
```

Listing 1: BFS og kø-implementasjon

Kompleksitet

Tidskompleksiteten er $O(N + M)$, ettersom hver node besøkes og hver kant behandles én gang. For en graf som lagres som `vector<vector<int» adj`, er det slik at lagring av alle naboer: $O(M)$ (én oppføring per kant), vektorhoder for N noder: $O(N)$, og tillegg fra BFS: `dist[N]`, `parent[N]` og køen gir inntil $O(N)$. Totalt plasskompleksitet er da $O(N + M)$.

Lesing og konstruksjon av grafen er $O(N + M)$. Å lese inn antall noder N og kanter M fra fil, samt alle kantene `(u, v)`, krever $O(N+M)$ tid, lineært i størrelsen på input. Å bygge nabolistene ved å legge til hver kant én gang koster $O(M)$ tid, slik at total bygging er $O(N + M)$.

Begrensninger

Det er fortsatt mange begrensninger i besvarelsen, som:

Filformat: Første linje i filen skal være `N M` (antall noder og kanter). Deretter følger nøyaktig `M` linjer med kanter `u v`. Programmet oppretter `adj(N)` og bruker `u` som indeks. Hvis `u` eller `v` er utenfor $[0, N-1]$, vil det oppstå indeksfeil. Besvarelsen kommer fra antagelsen at filen beskriver en rettet graf. For en urettet graf må begge retninger oppgis, eller legges til i koden

Uoppnåelige noder: Noder som ikke kan nås fra startnoden får `dist = -1` og `parent = -1`. I koden starter alle distanser som `INF` (uendelig stor avstand / ikke oppdaget ennå), og settes først når en node oppdages. Hvis dette aldri skjer er noden uoppnåelig. Ved utskrift vises slike felt som «-». Dette indikerer korrekt at ingen vei eksisterer.

Determinisme: BFS gir alltid samme avstander, men hvilken node som blir forgjenger (`parent`) kan variere dersom flere naboer oppdages samtidig. Køen (`queue`) er FIFO, rekkefølgen bestemmes av hvordan naboene ligger i `adj[u]`. For stabil og forutsigbar rekkefølge sorteres nabolistene før søket:

```
1 for(auto& nbrs : adj) sort(nbrs.rbegin(), nbrs.rend());
```

Listing 2: Rekkefølgen på utskriften (synkende)

Dette påvirker ikke avstandene, kun utskriftsrekkefølgen.

Skalerbarhet: `vector<vector<int>>` fungerer godt for små og mellomstore grafer. For svært store grafer er Compressed Sparse Row (CSR) mer effektiv. Denne metoden har mer kompakt minnebruk (kun to arrayer: `offsets` og `to`) (se i deloppgaven 2). Den har i tillegg bedre cache-lokalitet ved gjennomgang av naboer. CSR bør brukes når grafen har millioner av noder eller kanter, slik som i `Skandinavia.txt`.

Deloppgave 2: Topologisk sortering

I denne deloppgaven valgte jeg å bruke Kahns algoritme (BFS) for topologisk sortering og CSR for store grafer (`Skandinavia.txt`). Topologisk sortering finnes kun for DAGer (Directed Acyclic Graphs)/rettet, uten sykler. Grafen representeres i et komprimert rad-format (CSR) med to arrayer: `offsets` av størrelse $N+1$ og `to` av størrelse M. Under innlesing telles både inn- og ut-grader for hver node. Kahns algoritme starter med alle noder som har inn-grad 0, som legges i en kø. Deretter tas én node u ut av køen av gangen, legges til i resultatet, og alle utgående kanter $u \to v$ behandles ved å redusere

`indeg[v]` med 1 hvis den blir 0, legges v til i køen. Når alle noder er prosessert har vi en topologisk rekkefølge dersom ikke alle noder ble skrevet ut, betyr det at grafen inneholder en sykel (Ikke DAG).

```
vector<int> topo_kahn(const vector<int>& offsets, const
    vector<int>& to, int N, vector<int> indeg) {
    queue<int> q;
    for(int u = 0; u < N; u++) if(indeg[u] == 0) q.push(u);
    vector<int> order;
    order.reserve(N);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        order.push_back(u);
        for (int i = offsets[u]; i < offsets[u+1]; ++i) {
            int v = to[i];
            if (--indeg[v] == 0) q.push(v);
        }
    }
    if((int)order.size() != N) return {};   // cycle
    return order;
}
```

Listing 3: Kahns algoritme

Kompleksitet

Vi representerer grafen i et komprimert radformat (CSR) med to arrayer: `offsets` (størrelse N+1) og `to` (størrelse M). Bredde-først søk (BFS) prosesserer hver node og hver kant høyst én gang:

Tidskompleksitet er $O(N + M)$. Dette er fordi hver node settes i kø én gang, og for hver node traverseres nøyaktig intervallet `to[offsets[u]..offsets[u+1])`, som totalt summerer til M kanter. Plasskompleksitet er $O(N+M)$ for grafen (`offsets` og `to`), samt $O(N)$ tillegg for `dist`, `parent` og `queue`, dermed $O(N + M)$. Lesing av inngangsfilen er $O(N + M)$ (lineær i antall noder og kanter). CSR bygges ved prefikssummer ($O(N)$) og én passering over alle kanter ($O(M)$).

```
1  //  Teller utgrad/inngrad for a bygge CSR
2  vector<int> outdeg(N, 0), indeg(N, 0);
3      for(int i = 0; i < M; i++) {
4          int u, v;
5          if(!(file >> u >> v)) {
6              cout << "Forventet" << M << "kanter, men fant
                   f rre.\n";
7              return 1;
8          }
9          if(u < 0 || u >= N || v < 0 || v >= N) {
10             cout << "Kant utenfor omr de  p   linje " << (i
                   + 2) << ".\n";
11             return 1;
12         }
13         edges.emplace_back(u,v);
14         outdeg[u]++;
15         indeg[v]++;
16     }
17
18     // Bygger offsets med indekser fra CSR
19     vector<int> offsets(N+1, 0);
20     for(int u = 0; u < N; u++) offsets[u+1] = offsets[u] +
           outdeg[u];
21
22     vector<int> cursor = offsets;
23     vector<int> to(M);
24     for(const auto& e : edges) {
25         int u = e.first; int v = e.second;
26         to[cursor[u]++] = v;
27     }
28     edges.clear(); edges.shrink_to_fit();
```

Listing 4: Noen bruk av CSR i koden

Begrensninger

Filformat: Programmet forventer første linje `N M`, etterfulgt av M par `u v`. Noder må ligge i intervallet $[0, N-1]$. For urettet graf må begge retninger stå i filen.

Minne: CSR krever to arrayer av størrelsesorden N og M. For svært store grafer kan minne bli flaskehals. Vi bruker `int` for indekser; dersom N eller M kan overstige $2^{31}-1$, må 64-bits heltall (`int64_t`) vurderes.

Uoppnåelige noder: `dist[v] = -1` markerer at noden ikke nås fra start. Disse nodene får ingen forgjenger (`parent[v] = -1`).

Utskrift: Å skrive ut én linje per node kan bli tregt for store N. I praksis bør man begrense antall linjer som skrives (f.eks. de første 1000) eller skrive til fil.

Determinisme: BFS-avstander er entydige, men forgjenger kan variere dersom flere naboer oppdages i samme nivå. Med CSR og uendret kantrekkefølge blir resultatet deterministisk.

Retningssans: Grafen behandles som rettet. Hvis dataene logisk sett er urettede, må begge retninger være eksplisitt til stede, ellers blir avstandene feil.

Inndata-kvalitet: Programmet sjekker at alle kanter i filen er gyldige. Hver node-ID må ligge mellom `0` og `N-1`; hvis en kant peker til en node utenfor dette området, stoppes programmet med en feilmelding. Det kontrolleres også at filen faktisk inneholder alle de `M` kantene som er oppgitt på første linje. Dersom grafen bruker 1-baserte noder (starter fra 1 i stedet for 0), må ID-ene reduseres med 1 før grafen bygges.