HW2. Seletuskiri elektriga kauplemise algoritmi juurde

Õppeaine ITI0204 Algoritmid ja andmestruktuurid

Ülesande kirjeldus

Karl ja Anni on kaasaegsed tudengid: töötavad IT-valdkonnas, tihti teevad kaugtööd välismaal, neil on ka elektriauto Audio X. Nad otsustasid leiutada kontrolleri e. tarkvara, mis suudaks tunnihindade järgi valida millal energiat osta ja millal müüa, nendele tundidele vastavalt automaatselt akut häälestada ning seda nii, et saadud tulu oleks võimalikult suur. Aita Karlil ja Annil selline tarkvara luua. Äraviibimise ajaks aku on mingil määral laetud. Kui nad tulevad tagasi, ei tahaks nad veel oodata kuni auto laeb. Seega äraviibimise perioodi lõpuks peab aku ikka kindal tasemel laetud olema.

Teie ülesandeks on kirjutada algoritm, mis antud parameetrite järgi (tunnihinnad, aku mahtuvus, võrgu piirangud tunnis, võrgu kulu, eneria edastamise efektiivsus, aku algseis, aku lõppseis) valib sellise ostu- ja müügistrateegia, mis võimaldaks rohkem raha teenida. Börsireeglid näevad ette, et ühe tunni jooksul saab teha ainult ühte tehingut (st natuke osta ja natuke müüa sama tunni jooksul ei ole võimalik), aga seda ka ei pruugi teha (st saab ka oodata); ning tehingu suurus on alati täisarv (st 2.3 kWh osta/müüa ei saa, kuid 2 kWh või 3 kWh saab).
Sisend-väljund

Sisendiks on:

ühemõõtmeline massiv tunnihindadega senti/kWh (prices[])
aku mahtuvus kWh (capacitance). St ei saa osta rohkem kui on aku mahtuvus ning müüa rohkem kui akus on energiat
võrgu piirangud tunnis kWh (receivingLimit ja sendingLimit). St ühe tunni jooksul ei saa börsilt osta rohkem kui receivingLimit ega müüa (börsile saata) rohkem kui sendingLimit piirang ette näeb.
võrgu kulu senti/kWh (cost). St iga läbi võrgu edastatud kWh kohta tekib ka võrgu kulu.
eneria edastamise efektiivsus (transmissionEfficiency). St kui üks pool (aku/börs) saadab võrku x kWh, siis teise pooleni (vastavalt börs/aku) jõuab ainult round(x * transmissionEfficiency) kWh (ümardatud täisarv). Vt all kuidas see mõjutab arvutusi.
aku algseis kWh (initialLoad). St aku seis perioodi algul.
aku lõppseis kWh (finalLoad). St perioodi lõpuks peab aku olema laetud vähemalt sellisel tasemel (võib aga suurem olla).

Väljundiks on Result klassi objekt, millel on kaks meetodit. Result.getTransactions() meetod peab tagastama täisarvude listi, mille sisuks on tehingute jada, kus igale tunnile vastab üks tehing (positiivne arv tähendab ostutehingu suurust, 0 tähendab tehingust loobumist, negatiivne

arv tähendab müügitehingu suurust). Tehingu suurus on kasutaja päringu suurus, ehk palju ta börsilt küsib või üritab börsile saata (st eneria edastamise efektiivsust arvestamata). Result.getTotalIncome() tagastab valitud tehingute strateegiaga saadud kogutulu.

Eneria edastamise efektiivsus

Eneria edastamise efektiivsus näitab mis osa saadetud energiast jõuab saajani. Börs maksab ainult saadud energia eest ning võtab raha saadetud energia eest.

Nt. aku seis on 10, võrgu kulu on 0.1, eneria edastamise efektiivsus on 0.8, börsilt ostmise piirang (receivingLimit) on 4 ja börsile saatmise piirang (sendingLimit) on 3:

Kui ma ostan 4 kWh (rohkem ei saa), siis
börs võtab raha 4kWh eest
börs saadab võrku 4kWh
võrgu kulu on 4 * 0.1 = 0.4
akuni jõuab round(4 * 0.8) = round(3.2) = 3 kWh
Kui ma müün 3 kWh (rohkem ei saa), siis
aku saadab võrku 3kWh
võrgu kulu on 3 * 0.1 = 0.3
börsini jõuab round(3 * 0.8) = round(2.4) = 2 kWh
börs maksab 2kWh eest

Sisendandmete piirangud:

5 <= | prices |, e. tunnihindade massivi pikkus
10 <= capacitance
0.4 capacitance <= hourLimit
0.1 <= cost
0 < transmissionEfficiency <= 1 (lihtsamates testides eneria edastamise efektiivsus on 1)
0 <= initialLoad <= capacitance
0 <= finalLoad <= capacitance

Lahendus ei pea olema optimaalne. Mida lähemal on teie lahendus optimaalsele lahendusele, seda rohkem punkte saab. Tehingute massivi pikkus võib olla lühem kui hindade massivi pikkus. Nt. kui antud tunnihindade massivi pikkus on 100, aga teie saadate testrile ainult 70 (esimest) tehingut, tester tõlgendab seda nii, et ülejäänud tehingud on nullid (ehk te enam midagi ei tee).

NB! Kuigi lahendus ei pea olema optimaalne, ta peab olema ikkagi õige. St saadud tulu on õigesti arvutatud, tehingud on valitud vastavalt piirangutele (ei saa osta/müüa rohkem kui piirangud näevad ette) ning perioodi lõpuks aku seis ei tohi olla väiksem kui soovitud.

Tööprotsessi kirjeldus ja algoritmideni jõudmise protsess

Algoritmi leidmise olulisimaks piiranguks oli, et hindade hulk võis olla väga suur. Kõikvõimalike kombinatsioonide rekursiivne välja arvutamine ja võrdlemine ei olnud realistlik.
[…]
Kuna erinevate tehingute kaalumine järjestikustel ajaperioodidel on optimaalse kombinatsiooni leidmine, siis teise lahendusena tekkis mõte, et seda oleks sobilik lahendada **dünaamilise planeerimise algoritmiga**. Keerukas oli dünaamilise planeerimise minimaalse ülesande leidmine: selge oli, et **massiivi üheks mõõtmeks on aeg**, mis pidevalt suureneb. Kuid aku mahu lisamine minimaalsesse ülesandesse oli keerukam. Sai katsetatud ka **massiivi teise mõõtmena** ostu-müügi mahu lisamist (st vahemikus negatiivne maksimaalne müügikogus -> 0 -> maksimaalne ostukogus), kuid see osutus väga keerukaks, sest tegelik aku laetus muutus hoopiski sõltuvalt ülekande efektiivsusest. Vahepealse lahendusena sai proovitud Floydi algoritmi põhist lähenemist, kuid see osutus äärmiselt ebaefektiivseks, juba aastast lühemate perioodidega tekkisid mälu piiratuse ja timeOut tõrked. Viimase ja ja lõpliku lahendusena sai lisatud **massiivi teiseks mõõtmeks aku mahutuvus** (st vahemikus 0 kuni capacity). Lõplikus algoritmis peab DP massiiv iga tunni ja laetuse taseme kohta meeles parimat võimalikku kasumit.
[...]

2. Teine lahendus. Meetod optimalTransactionsWithDP.

2.1. Lahendusmeetodi ja algoritmi kirjeldus

Tegu on dünaamilise planeerimise algoritmiga. **Esmalt luuakse kahemõõtmeline massiiv dp[hour][load]**, kus tunnid on vahemikus 0 kuni timeSteps ja load on vahemikus 0 kuni aku mahtuvus (capacitance). See massiiv hoiab iga tunnisammu lõpuks **suurimat võimalikku kasumit antud aku laetustasemega**. Teiseks massiiviks on valitud tegevusi (st ostu-müügi koguseid) hoidev **massiiv transaction[hour][load]**.
Esimesel tunnil omistatakse kohal dp[0][initialLoad] kasumiks 0 ja kõikil teistel laetuse tasemetel negatiivne lõpmatus. See tagab, et tehingute alguspunktina kasutatakse algset laetust. Iga tunni jaoks kaalutakse kolme põhilist tegevust: mitte midagi teha, energiat osta (kuni receivingLimit) või energiat müüa (kuni sendingLimit). Ostmisel ja müügil arvestatakse võrgukulusid ning energia ülekande efektiivsust (transmissionEfficiency). Iga võimalik tehingu tulemus arvutatakse läbi ja **võrreldakse dp tabelis vastava järgmise laetuse taseme seisuga dp[hour+1][newLoad]**. Kui see annab parema kasumi, siis uuendatakse dp-tabelit ning salvestatakse vastav tegevus transaction massiivi.
Protsessi **lõpuks vaadatakse dp[timeSteps][finalLoad]**, et leida maksimaalne kasum, mille juures aku laetuse tase on nõutaval aku lõppseisu (finalLoad) tasemel. Sellest punktist **liigutakse sammhaaval tagasi** (sh ostutehingu korral arvestatakse, et tegelik muutus laetuse tasemes on energia ülekande kao tõttu on madalam kui ostetud kogus) ja leitakse kogu optimaalne tegevuste järjekord (resultTransactions). **Kasum arvutatakse tegevuste järgi** sammhaaval eraldi meetodiga calculateProfitFromTransactions().

2.2. Miks lahendus on korrektne

Lahendus on korrektne, sest tehingute kasumit uuendatakse massiivis ainult sel juhul, kui see vastab **aku mahutavuse piirang**ule. Igal sammut võetakse arvesse energia ülekande efektiivsust ja võrgu kulu, ostu korral liigutakse DP massiivis edasi arvestades **tegelikku muutust laetuse tasemes**, mitte arvestades ostetud kogust. **Aku algseisu** arvestatakse omistusega dp[0][initialLoad] = 0. **Aku lõppseisu tingimus** täidetakse sellega, et tegevuste järjekord koostatakse nö tagurpidi ja viimaseks tegevuseks valitakse massiivi positsioon [timeSteps][finalLoad], st viimase tunni nõutav lõpliku laetuse tase.

2.3. Asümptootilise keerukuse hinnang ja põhjendus

Kui n on hindade (päevade) arv, siis kahemõõtmeline dünaamilise planeerimise tabel on suurusega n x aku mahutuvus. Igal (hour, load) olekul kaalutakse kuni receivingLimit + sendingLimit + 1(st mitte midagi tegemine) tegevust. Kui lubatud tegevust vahemik = receivingLimit + sendingLimit +1 on algoritmi keerukuseks **O(n * aku mahutuvus * lubatud tegevuste vahemik)**.

3. Lahendusmeetodite ja algoritmide võrdlus

3.1.Sisuline võrdlus

[…]
Teine lahendus on **dünaamilise planeerimise algoritm**. Selle lahenduse põhiliseks puuduseks on **suurem ajaline keerukus**: koos sisendi suurenemisega suurenevad ka dünaamilise planeerimise otsuseid toetavad massiivid. Suureks eeliseks on laiemat pilti arvestavad optimaalsemad tehingud, mis suudavad hästi kasumisse jääda.

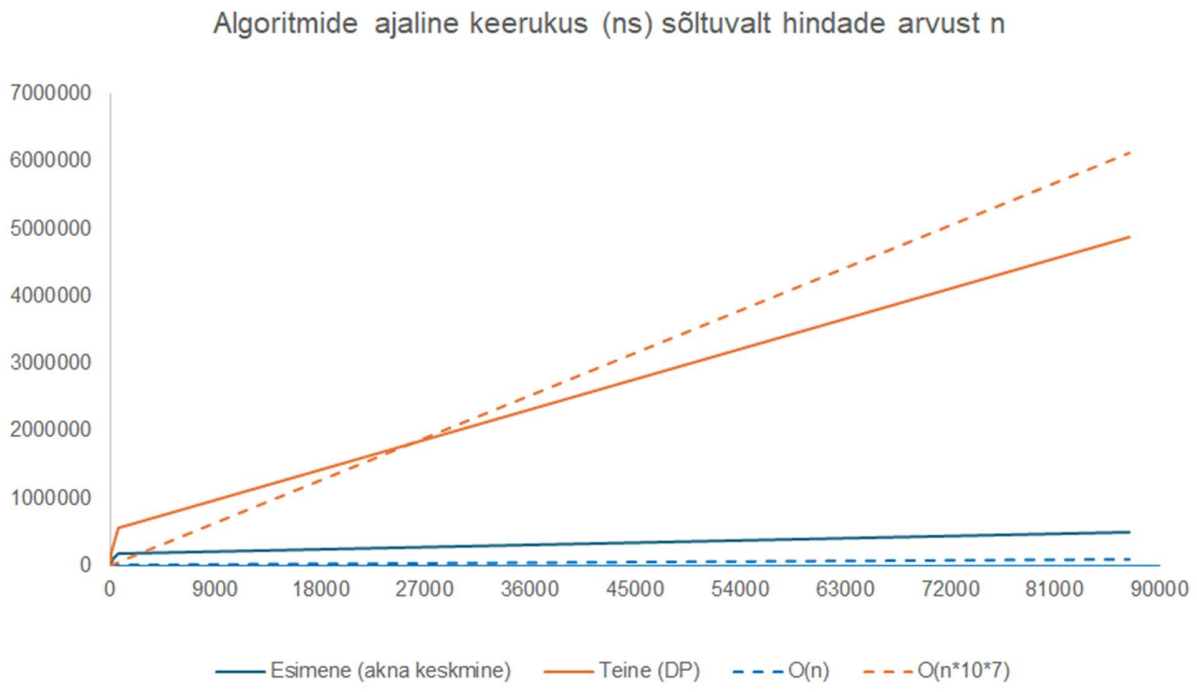
3.2. Katseline võrdlus

Katselise võrdluse tegin arvestades algoritmi läbikäimiseks kulunud aega nanosekundites Java nanoTime() meetodiga. Katseline võrdlus kinnitas algoritmi keerukuse hinnangut: esimene lahendus on märkimisväärselt madalama ajalise keerukusega (Tabel 1).

Tabel 1. Esimese ja teise lahenduse ajalise keerukuse katseline võrdlus. Näidatud ajakulu on 1000 meetodi väljakutse keskmine. Lisaks ajakulule nanosekundites on näidatud ajakulu jagatuna hindade arvuga n.

	Esimene lahendus		**Teine lahendus**	
	ajakulu (ns)	(aeg/n)	ajakulu (ns)	(aeg/n)
Üks päev (n=24)	46061	1919	51368	2140
Kaks päeva (n=48)	31258	651	89296	1860
Kolm päeva (n=72)	41077	570	180579	2508
Üks nädal (n=168)	87368	520	273454	1627
Üks kuu (n=720)	189051	281	559127	832
Aasta (n=87369)	500774	57	4871056	557

Mõõtmiste tulemuste kandmine graafikule kinnitas, et esimese ja teise algoritmi vastavad hinnangulised keerukused O(n) ja O(n * aku mahutuvus = 10 * lubatud tegevuste vahemik = 7) on väga lähedased katseliselt saadud tulemustele (Joonis 1).



Joonis 1. Esimese ja teise algoritmi ajaline keerukus nanosekundites (püst-teljel) sõltuvalt hindade arvust (horisontaalteljel). Punktiirjoonega on kujutatud vastavad keerukuse hinnangud O(n) ja O(n * aku mahutuvus * lubatud tegevuste vahemik).