Fakultet elektrotehnike i računarstva
Zavod za primjenjeno računarstvo

Napredni algoritmi i strukture podataka

2. laboratorijska vježba

Filip Voska 0036467446

Zagreb, 5. prosinca 2014.

1. Zadatak

Zadatci za 11 (17) bodova

Napisati program (poželjno C, C++, C# ili Java) koji će dinamičkom strategijom (dinamičkim programiranjem) riješiti problem u kojem svaka kategorija ima konačni broj podkategorija, a može se uzeti samo po jedan član tih podskupova. U primjeru s predavanja, to bi značilo da npr. postoje iste stvari od različitih materijala pa su im različite i vrijednosti i težine, a lopov može (želi) uzeti najviše jednu stvar „svake vrste“, pri čemu je ograničenje ukupna težina, a ne volumen.

- za kolokviranje vježbe važno je pregledno i jasno opisati problem i njegove karakteristike koje ga čine prikladnim za primjenu dinamičkog programiranja,
- naglasak rada treba biti na kvalitetnom programu pa u dokumentaciji treba jasno (ali sažeto!) prikazati ideju rješenja i organizaciju programa (strukture podataka, pomoćne funkcije, itd.). Programski kod ne upisivati u dokumentaciju, osim kratkih izvadaka ako se radi o posebno važnim blokovima.

2. Rješenje zadatka

2.1. Teorijski uvod

Dinamičko programiranje

Dinamičko programiranje jer tehnika rješavanja problema koje je moguće podijeliti u manje cjeline koje se mogu zasebno rješavati. Princip algoritama baziranih na dinamičkom programiranju se osniva na rješavanju manjih i jednostavnijih dijelova programa koji nisu zahtjevni. Izračunati dijelovi rješenja se koriste kako bi se konstruiralo cjelokupno rješenje problema.

Dinamičko programiranje je pogodno za rješavanje preklapajućih potproblema te određivanja optimalnih podstruktura. Često je korisno prilikom rješavanja rekurzivnih problema kao što je izračun brojeba Fibonaccijevog niza, a često se koristi i u disciplinama kao što je Bioinformatika (i općenito u algoritmima nad nizovima), gdje se primjerice koristi za određivanje sličnosti nizova.

Jedan od najčešćih primjera problema koji se rješavaju dinamičkim programiranjem je takozvani Knapsack problem – problem u kojemu imamo ograničene resurse te određeni broj predmeta koji imaju određene vrijednosti. Svaki predmet možemo odabrati najviše jednom, pri čemu trošimo resurse, a povećavamo ukupno dobivenu vrijednost. Cilj je odabrati takvu kombinaciju predmeta koja će maksimizirati dobivenu vrijednost za ograničene resurse.

U proširenoj verziji Knapsack problema postoji mogućnost grupiranja predmeta na način da se iz jedne grupe predmeta može odabrati samo jedan predmet. Ukoliko se grupa sastoji od više podgrupa, prvo se riješe pojedinačne podgrupe te se njihova rješenja koriste za rješavanje grupe unutar koje se nalaze. To se može postići

postavljanjem više Knapsack problema za pojedine podgrupe te samo po sebi ne mijenja algoritam, već samo znači da se algoritam mora izvesti više puta za razne podgrupe. Podgrupe u tom slučaju predstavljaju pojedinačne Knapsack probleme te se ovim postupkom problem svodi na odabir predmeta iz grupa pri čemu grupa sadrži samo osnovne predmete, a ne i druge grupe. U ovoj laboratorijskoj vježbi sam odlučio rješavati osnovni problem u kojemu imamo određeni broj grupa po kojemu rasporedimo predmete, a za problem podgrupa unutar grupa se algoritam treba izvesti nekoliko puta.

Konkretni problem

Problem koji se rješava je odabir raketnih pogona za letjelicu (zrakoplov, space shuttle, raketa ili nešto treće). Postoji više proizvođača pogona (npr. Boeing, Airbus, Lockheed Martin, …) te svaki od tih proizvođača nudi određeni asortiman pogona. Proizvođači predstavljaju grupe, a pogoni predmete. Svaki pogon ima određenu snagu (vrijednost) te cijenu. Cilj je uz ograničeni budžet odabrati pogone koji će pružati najviše zajedničke snage. Pritom je važno naglasiti da zbog zakonskih regulativa (konkurencija tržišta) od svakog proizvođača smijemo kupiti najviše jedan pogon.

Tablica 1 - Podatci o pogonima za konkretni problem

Pogon	Snaga	Cijena	Grupa
RB-211	10	8	Boeing
Jet A-1	5	3	Boeing
X-32	20	10	Boeing
E3000	5	3	Airbus
PW127G	4	2	Airbus
V-280	3	2	Lockheed Martin
P-3	1	1	Lockheed Martin

*napomena: vrijednosti snage i cijene ne odgovaraju stvarnim vrijednostima iz života.

Problem je prikladan za rješavanje korištenjem dinamičkog programiranja jer predstavlja problem odabira predmeta s vrijednošću i cijenom uz ograničene resurse – Knapsack problem uz grupe (bez podgrupa).

Nakon pokretanja programa uz ograničenje ukupne cijene postavljeno na 15, rezultat je sljedeći:

- odabrani pogoni: V-280, X-32, E3000
- ukupna vrijednost: 28,
- ukupna cijena: 15,
- ostatak: 0

Način izračuna optimalnog rješenja se može izvesti na dva osnovna pristupa:

1. Generirati Knapsack probleme koji neće sadržati grupe, već samo pojedinačne predmete. Ovaj pristup znači da iz svake grupe moramo odabrati jedan predmet te tako odabrane predmete riješiti kao klasični Knapsack problem. Tako generiramo velikog broj klasičnih Knapsack problema koji se moraju riješiti te na kraju usporediti kako bi se ustanovilo koji odabir predmeta iz kojih grupa je optimalan. To znači da će se za n grupa od kojih svaka ima m predmeta generirati m^n Knapsack problema koje treba riješiti. U svakom pojedinačnom Knapsack problemu će se nalaziti n predmeta (iz svake grupe po jedan), a svaki od predmeta može biti odabran na m načina. Očito je da za veliki broj grupa s velikim bojem predmeta imamo eksploziju vremenske složenosti koja je eksponencijalna. Prostorna složenost se može održati kvadratnom, no trebat ćemo imati barem 2 tablice u svakom trenutku – jedna koju trenutno obrađujemo te jedna koja pamti dosada najbolje rješenje.

2. Drugi pristup je zadržati problem u jednoj tablici, ali podijeliti predmete u grupe. To znači da ćemo morati modificirati klasični Knapsack algoritam. Klasični Knapsack algoritam za izračun vrijednosti ćelije mora znati vrijednosti ćelija prethodnog predmeta. Ono što moramo promijeniti je da se provjeravaju vrijednosti ne prethodnog predmeta, već svih predmeta iz prethodne grupe. Nije dopušteno unutar grupe računati vrijednosti ćelije temeljem predmeta iz iste grupe, već je potrebno iskoračiti u prethodnu grupu. Vremenska složenost u ovom pristupu nije eksponencijalna kao u prvom, već se smanjila na kvadratnu te ovisi o broju predmeta (neovisno o broju grupa) te o ograničenju resursa. Prostorna složenost je također kvadratna kao i u prvom pristupu, pošto i dalje moramo pamtiti cijelu tablicu kako bismo se znali vratiti unatrag i utvrditi koji predmeti su odabrani.

Oba pristupa daju ispravne rezultate. Prvi je zahtjevniji što se tiče procesorske snage, a drugi je nešto kompliciraniji za implementaciju. Odabran je drugi pristup za implementaciju rješavanja problema, a kako bi se dodatno pojasnila razlika u pristupima, pogledajmo primjer (iz prezentacije) za sljedeće predmete:

Predmet	Vrijednost	Cijena	Grupa
i1	8	5	g1
i2	6	2	g2
i3	12	7	g2
i4	5	3	g3

Maksimalna ukupna cijena: 12.

Slika 1 - Knapsack bez grupa prikazuje rezultat algoritma ako ne radimo s grupama – rezultat se poklapa s rezultatom primjera iz predavanja, uzimamo predmete 2, 3 i 4 koji zajedno stoje 12 i vrijede 23.

Items / Cost levels	i1	i2	i3	i4
1	0	0	0	0
2	0	6	6	6
3	0	6	6	6
4	0	6	6	6
5	8	8	8	11
6	8	8	8	11
7	8	14	14	14
8	8	14	14	14
9	8	14	18	18
10	8	14	18	19
11	8	14	18	19
12	8	14	20	23

Take Items:

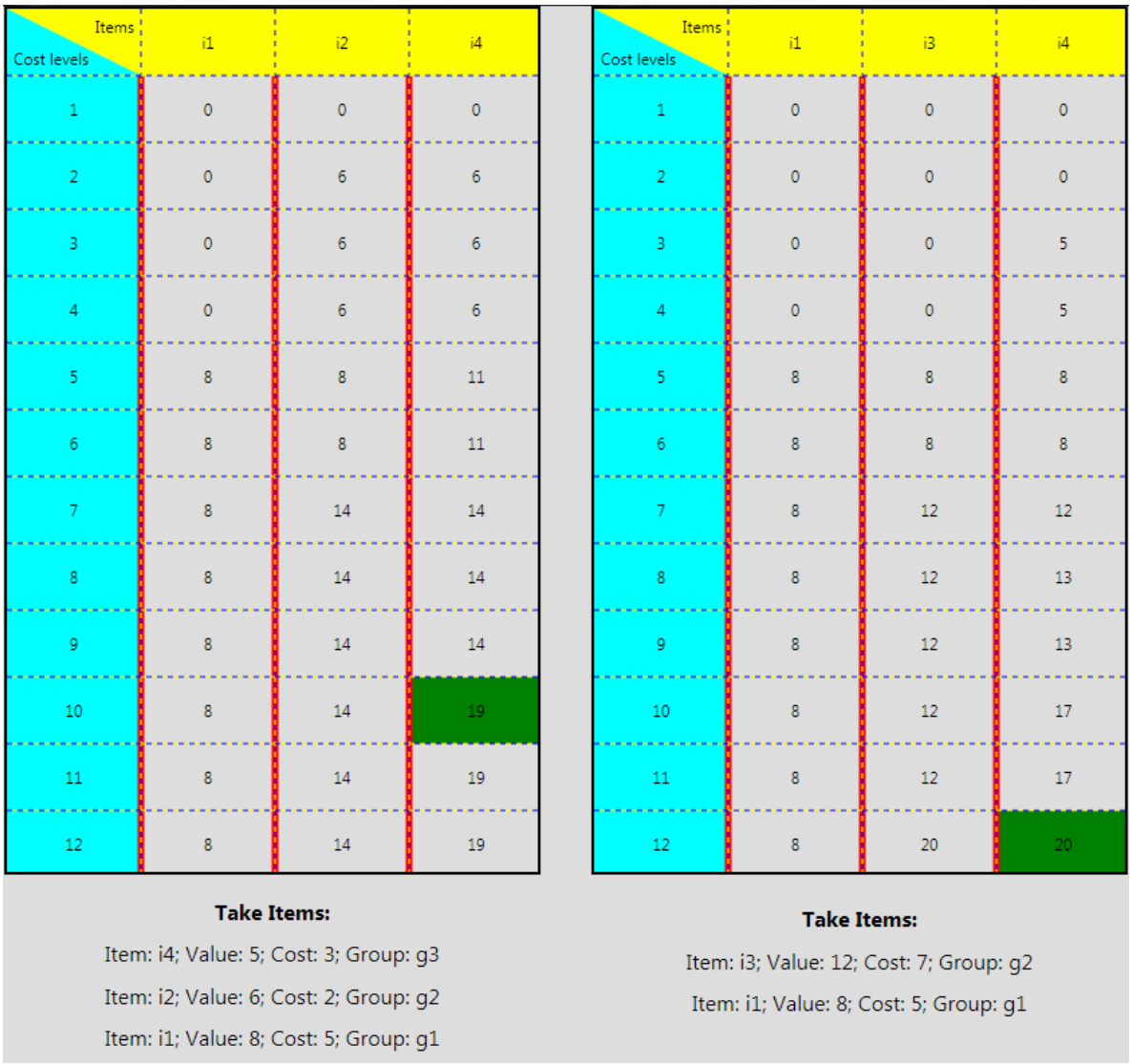
Item: i4; Value: 5; Cost: 3; Group: n/a

Item: i3; Value: 12; Cost: 7; Group: n/a

Item: i2; Value: 6; Cost: 2; Group: n/a

Slika 1 - Knapsack bez grupa

Slika 2 - Knapsack s grupama - 1. pristup prikazuje kako bi program radio kada bi se koristio 1. pristup. Primjer je maleni pa bi trebalo izračunati samo 2 osnovna Knapsack problema, no jasno je da broj osnovnih problema raste eksponencijalno s brojem brojem predmeta u grupama. Samo jedna grupa ima više od 2 predmeta – grupa 2 (i2 i i3). To znači da ćemo prvo morati riješiti osnovni problem za predmete i1, i2 i i4, a zatim za predmete i1, i3 i i4. Kada riješimo oba, usporedimo optimalna rješenja te uzimamo ono koje je veće – u ovom primjeru to je slučaj kada uzmemo i3 i i4 koji imaju vrijednost 20 (uz cijenu 12 i ostatak 0), naspram i1, i2 i i4 koji imaju vrijednost 19 (uz cijenu 10 i ostatak 2).



Items / Cost levels	i1	i2	i4
1	0	0	0
2	0	6	6
3	0	6	6
4	0	6	6
5	8	8	11
6	8	8	11
7	8	14	14
8	8	14	14
9	8	14	14
10	8	14	19
11	8	14	19
12	8	14	19

Take Items:

Item: i4; Value: 5; Cost: 3; Group: g3

Item: i2; Value: 6; Cost: 2; Group: g2

Item: i1; Value: 8; Cost: 5; Group: g1

Items / Cost levels	i1	i3	i4
1	0	0	0
2	0	0	0
3	0	0	5
4	0	0	5
5	8	8	8
6	8	8	8
7	8	12	12
8	8	12	13
9	8	12	13
10	8	12	17
11	8	12	17
12	8	20	20

Take Items:

Item: i3; Value: 12; Cost: 7; Group: g2

Item: i1; Value: 8; Cost: 5; Group: g1

Slika 2 - Knapsack s grupama - 1. pristup

Slika 3 - Knapsack s grupama - 2. pristup prikazuje kako bi program radio za 2. pristup, a to je ujedno i pristup koji je implementiran. Za razliku od 1. pristupa, vidimo da imamo samo jednu tablicu te imamo razdijeljene grupe (crvene crte). Važno je uočiti da ćelija [2, i3] ima vrijednost 0, a ne 6. To je zato što ne smije preuzeti lijevu vrijednost od predmeta koji se nalazi u istoj grupi. Međutim, predmeti i1 i i2 nisu u istoj grupi te ćelija [5, i2] može poprimiti vrijednost iz lijeve grupe. Spomenimo još i ćeliju [7, i4] koja je poprimila vrijednost 14 – maksimalnu vrijednost za isto cjenovno ograničenje (7) u prethodnoj grupi. Uočimo da se nije preuzela vrijednost 12, već 14 – traži se maksimum unutar grupe, a ne preuzima se direktno iz lijevog stupca.

Items / Cost levels	i1	i2	i3	i4
1	0	0	0	0
2	0	6	0	6
3	0	6	0	6
4	0	6	0	6
5	8	8	8	11
6	8	8	8	11
7	8	14	12	14
8	8	14	12	14
9	8	14	12	14
10	8	14	12	19
11	8	14	12	19
12	8	14	20	20

Take Items:

Item: i3; Value: 12; Cost: 7; Group: g2

Item: i1; Value: 8; Cost: 5; Group: g1

Slika 3 - Knapsack s grupama - 2. pristup

2.2. Implementacija

Problem je riješen u programskom jeziku C# te je razdvojen u dva projekta – Knapsack Solver te Knapsack Client. Implementiran je u prethodnom poglavlju navedeni 2. pristup. Knapsack Solver sadrži svu logiku potrebnu za izračun optimalnog rješenja te se može koristiti kao vanjska biblioteka, dok Knapsack Client pruža korisničko grafičko sučelje od Knapsack Solvera prema korisniku. Grafičko sučelje je izgrađeno koristeći Windows Presentation Foundation. Izgrađena je Desktop aplikacija, no moguće je izraditi i Web aplikaciju koristeći Knapsack Solver.

2.1.1. Strukture podataka

Osnovna struktura u Knapsack Solveru je tablica u koju se spremaju rezultati izračuna vrijednosti ćelija. Broj stupaca je određen brojem predmeta, a broj redaka je određen razinama resursa, tj. maksimalnom dozvoljenom ukupnom cijenom. Prema klijentima je otvorena i lista predmeta koji se nalaze u optimalnom rješenju kako bi se olakšao dohvat rješenja. Važno je naglasiti da je za implementaciju odabranog pristupa bitno da su predmeti iste grupe zaista i „grupirani“ zajedno, tj. da se između njih ne nalaze predmeti druge grupe. Zato je potrebno sortirati predmete po grupi prilikom inicijalizacije tablice. Sortiranje bi se moglo izostaviti ukoliko bi klijenti garantirali da će predmete unositi redoslijedom određenim grupama.

2.1.2. Izračun vrijednosti ćelije

Za izračun vrijednosti ćelije moramo najprije saznati gdje se predmet nalazi u trenutnoj grupi te koliko je predmeta u grupi prije njega.

Položaj predmeta unutar njegove grupe određuje koliko stupaca ulijevo moramo gledati, jer prilikom izračuna vrijednosti ćelije ne smijemo uzimati u obzir predmete iz njegove grupe pošto smijemo odabrati najviše jedan predmet iz svake grupe.

Broj predmeta u prethodnoj grupi je bitan za traženje optimalnog rješenja iz prethodnog koraka. U klasičnom Knapsack problemu smo gledali ćeliju lijevo i ćeliju „dijagonalno“ (određenu razlikom trenutnog ograničenja te cijene trenutnog predmeta). Ovdje također gledamo lijevo i dijagonalno, no moramo gledati sve predmete u prethodnoj grupi. Ako prethodna grupa ima n predmeta, to znači da treba naći najveću vrijednost u n lijevih predmeta te najveću vrijednost u n „dijagonalnih“ predmeta. Zatim odredimo je li optimalno rješenje došlo iz lijevog ili dijagonalnog smjera (tražimo maksimum) te nam to određuje vrijednost ćelije te možemo označiti otkuda smo došli.

2.1.3. Određivanje odabranih predmeta

Određivanje odabranih predmeta je veoma slično klasičnom Knapsack problemu. Razlika je u tome što optimalno rješenje ne predstavlja nužno vrijednost u zadnjem retku i zadnjem stupcu, već se maksimalna vrijednost može nalaziti bilo gdje u zadnjem retku. Razlog tome je što ne znamo koji predmet iz zadnje grupe je odabran. Moguće je da zadnja grupa ima primjerice 3 predmeta te da je iz te grupe

odabran 2. predmet – to znači da će u tom slučaju najveća vrijednost biti u ćeliji zadnjeg retka i predzadnjeg stupca.

Također valja voditi računa o tome da možda nismo uspjeli doseći limit cijene predmeta. Pretpostavimo da je optimum u predzadnjem stupcu. Ukoliko je suma cijena odabranih predmeta manja od maksimalne zajedničke cijene za x, to znači da će se najveća vrijednost nalaziti u zadnjem retku i predzadnjem stupci, ali isto tako i u predzadnjem stupcu i u x-zadnjih redaka. U tom slučaju imamo „višak“ sredstava, no to ne predstavlja neki osobiti problem, već samo znači da nismo mogli potrošiti sve resurse jer su neki predmeti možda bili preskupi, njihovim odabirmo bismo dobili manju ukupnu vrijednost, a jeftinijih predmeta koji bi stali u cijenu viška više nije bilo dostupnih.

2.1.4. Pomoćne funkcije

Implementirane su razne male pomoćne funkcije za dohvat ćelija, grupa i vrijednosti koje provjeravaju rubne uvjete.

2.1.5. Grafičko sučelje

Prozor sučelja je podijeljen u dva dijela – lijevi i desni.
Lijevi dio pruža akcije s sljedećim mogućnostima:

- unos novog predmeta uz odgovarajući opis, vrijednost, cijenu te grupu. Moguće je rješavati i obične Knapsack probleme jednostavno ne unoseći grupu ni za jedan predmet,
- pokretanje izračuna,
- reset podataka i rješenja,
- učitavanje i rješenje problema s odabirom pogona za letjelicu.
    - ovaj problem je opisan u drugom poglavlju, a prikaz rješenja se može vidjeti pokretanjem programa. Slika tablice rješenja je prevelika za prikaz u ovom dokumentu.

Desni dio prikazuje rješenje problema - tablicu s popunjenim vrijednostima ćelija te ispod tablice popis predmeta koji su odabrani. Prelaskom miša preko ćelije ispisuju se podatci o predmetu koji odgovara toj ćeliji. Grupe su razdvojene vertikalnim crvenim linijama. Optimalno rješenje je označeno zelenom pozadinom ćelije.

3. Zaključak

Dinamičko programiranje je način rješavanja određene skupine problema koji se mogu razdijeliti na manje probleme.

Problem koji se rješavao je sadržavao više predmeta koji su se nalazili unutar neke grupe. Prilikom odabira predmeta iz svake grupe moguće je odabrati najviše jedan predmet. Postoji više načina na koji se problemu može pristupiti. Predložena su dva načina rješavanja problema s prednostima i manama te je temeljem toga izvršen odabir načina rješavanja koji je konačno implementiran programski.

4. Literatura

[1] Hlupić N., Kalpić D., FER, prezentacija „Dinamičko programiranje“
[2] Knapsack problem, Wikipedia, http://en.wikipedia.org/wiki/Knapsack_problem, datum posjete: 5.12.2014.