Fakultet elektrotehnike i računarstva Zavod za primjenjeno računarstvo

Napredni algoritmi i strukture podataka

2. laboratorijska vježba

Antonia Elek, 0036477925

Zagreb, 7.12.2016.

1. Zadatak

Zadatak za 11 bodova
Napisati program (poželjno C, C++, C# ili Java) koji će dinamičkom strategijom (dinamičkim programiranjem) riješiti problem u kojem svaka kategorija ima konačni broj potkategorija, 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.

2. Rješenje zadatka

2.1. Teorijski uvod

Dinamičko programiranje je strategija (metoda) kojoj je osnovno načelo postupno graditi rješenje složenog problema koristeći rješenja istovrsnih manje složenih problema. Ova strategija je primjenjiva kada se potproblemi “preklapaju” – za razliku od podijeli pa vladaj strategije koja problem rješava top-down pristupom, ne ponavlja već obavljeni posao jer maksimalno iskorištava rezultate prethodnih koraka. [1]

2.2. Knapsack problem

Problem koji je potrebno riješiti je odabir najbolje kombinacije stvari koje možemo kupiti, ako na raspolaganju imamo ograničenu količinu novca. U konkretnom primjeru, raspolažemo s 90 jedinca neke valute, želimo kupiti po jedan album i jednu knjigu, a na raspolaganju nam se nalaze dva albuma i tri knjige. Svojstva koja ovaj problem čine prikladnim za rješavanje dinamičkim programiranjem su optimalna podstruktura (za pronalaženje najboljeg rješenje koriste se najbolja rješenja nezavisnih potproblema) i međusobno preklopljeni potproblemi (prethodni rezultati se mogu iskoristiti za rješavanje kasnijih koraka).

Element	Kategorija	Vrijednost	Cijena
Rattle And Hum	Albumi	3	30
A Hard Day’s Night	Albumi	7	60
1984	Knjige	4	50
Anna Karenina	Knjige	3	50
The Bourne Identity	Knjige	5	40

Rješavanje ovog problema Knapsack algoritmom započinjemo izradom tablice svih mogućnosti koje možemo izabrati, i za svaku kombinaciju računamo vrijednost po formuli:
`max{ max(vk-1, c), max(vk-1, c-this.cost) + value(k) }`

2.3. Implementacija

Implementacija je ostvarena u programskom jeziku C# i se sastoji od klasa `Element`, `Category` i `Knapsack` kojima se modelira element sa svojstvima vrijednosti i cijene i koji je osnovni gradivni element kategorije i struktura knapsack problema koji se sastoji od više kaktegorija s pripadajućim elementima.

2.3.1. Element

Klasom Element modeliran je osnovni element Knapsack algoritma koji sadrži svojstva cijena, vrijednost i ime.

2.3.2. Kategorija

Klasom Category modelirana je kategorija elemenata od kojih je samo jednog (ili niti jednog) moguće izabrati u konačnom rješenju problema. Osim kolekcije elemenata, kategorija još ima i svoje ime.

2.3.3. Tablica rješenja

Knapsack tablica modelirana je u okviru klase `Knapsack` kao javno svojstvo tipa `Dictionary<KeyValuePair<decimal, Element>, decimal>`. Ključ ovog rječnika predstavlja ćeliju u tablici koja se nalazi na križanju određene cijene (redak) i određene stvari (stupac) pa je modeliran odgovarajućim uređenim parom `KeyValuePair<decimal, Element>`. Pripadna vrijednost svakog ključa predstavlja vrijednost koja je upisana u tu ćeliju. Također, za svaku kombinaciju (ćeliju) se u rječniku tipa `Dictionary<KeyValuePair<decimal, Element>, bool>` pamti ulazi li trenutni element u optimalno rješenje.

2.3.4. Popunjavanje tablice

Najprije je potrebno izračunati sve moguće cijene koje je moguće dobiti kombinacijom cijena elemenata. Moguće je ograničiti kombinacije nekom maksimalnom cijenom (90 u primjeru), a ako se ona ne navede, algoritam će računati za sve moguće kombinacije cijena stvari bez ograničenja. Moguće ukupne cijene se pohranjuju u kolekciju Costs unutar klase Knapsack. Tablica se popunjava prolaskom po svim elementima i po svim ukupnim cijenama koje se mogu postići i izračunom vrijednosti po formuli :

$$\max\{ \max(v_{k-1}, c), \max(v_{k-1}, c-this.cost) + value(k) \}$$

Tu je `max(vk-1, c)` maksimalana vrijednost koju je moguće postići u prethodnoj kategoriji za trenutnu sumu cijena, a `max(vk-1, c-this.cost) + value(k)` maksimalana vrijednost koju je moguće postići u prethodnoj kategoriji za sumu cijena umanjenu za cijenu trenutne stvari, plus vrijednost trenutne stvari.

Popunjena tablica:

Kategorija	Albumi		Knjige		
Element Cijena	Rattle and Hum	A Hard Day's Night	1984	Anna Karenina	The Bourne Identity
30	3	0	3	3	3
40	3	0	3	3	5
50	3	0	4	3	5
60	3	7	7	7	7
70	3	7	7	7	8
80	3	7	7	7	8
90	3	7	7	7	8

2.3.5. Rješenje

Za izračunavanje liste stvari koje čine optimalno rješenje najprije se pronalazi najveća vrijednost u tablici, a koja pripada zadnjoj izračunatoj kategoriji i najvećem retku (koji odgovara kapacitetu), što je ujedno i maksimalna vrijednost koju je moguće postići. Iščitava se za koju stvar se postiže ta vrijednost i provjerava se je li t stvar u optimalnom rješenju i ako jest, potom se od te vrijednosti oduzima vrijednost izabrane stvari i ostatak postavlja za novu maksimalnu vrijednost. Izabrana stvar se pohranjuje u listu izabranih stvari a postupak kreće ispočetka, s tim da se sada traži stvar u prethodnoj kategoriji. Ta petlja se vrti dok god maksimalna vrijednost koja se može ostvariti ne bude 0 ili dok se ne prođe kroz sve kategorije. U konkretnom primjeru, maksimalna vrijednost koju je moguće ostvariti iznosi 8, a postiže se za knjigu The Bourne Identity i album Rattle and Hum.

3. Zaključak

Jedna od najboljih karakteristika dinamičkog programiranja jest obrađivanje problema bottom-up pristupom, gdje se kreće od izračunavanja rješenja najjednostavnijih potproblema, koja se potom ugrađuju u rješenja kompleksnijih potproblema koji slijede i u konačno rješenje čitavog problema.

4. Literatura

[1] Nikica Hlupić i Damir Kalpić, Dinamičko programiranje, FER, http://www.fer.unizg.hr/_download/repository/Dinamicko_programiranje.pdf