Милан Чабаркапа

УВОД У ДИНАМИЧКО ПРОГРАМИРАЊЕ

1. Уводни део часа

Веома важна особина већине алгоритама је њихова ефективност. Ако је алгоритам такав да се до решења долази веома споро употребом алгоритма „грубе силе“ и/или захтева много меморије или неких других рачунарских ресурса, онда он врло често у примени може бити неупотребљив. Метода *динамичког програмирања* ефикасно доводи до решења проблема поступком који је сличан стратегији „подели па владај“, али уместо да се исти потпроблем решава више пута, што је често присутно код ове стратегије, прво решење потпроблема се сачува и користи по потреби.

Вишеструко решавање истог потпроблема није једини критеријум који захтева примену методе динамичког програмирања. Други важан критеријум за примену ове методе је својство да оптимално решење основног проблема представља комбинацију оптималних решења његових потпроблема. Како искомбиновати оптимална решења потпроблема обично је један од тежих задатака у току примене ове технике.

Идеја методе динамичког програмирања се састоји у свођењу (коришћењем рекурентних веза) полазног проблема на његове потпроблеме „мањих димензија“ и коришћење технике табелирања за чување добијених резултата. Основни циљ је избећи рачунање исте „ствари“ више пута, тако што би се формирала табела коју попуњавамо почев од резултата који најлакше добијамо и које користимо за наредна сложенија израчунавања. Динамичко програмирање је техника „одоздо на горе“, јер се обично стартује од најједноставнијих проблема, чијим се комбиновањем долази до решења сложенијих проблема, све до коначног решења задатог проблема.

У следећим задацима биће илустрован поступак решавања проблема применом методе динамичког програмирања и дате идеје за решавање њима сличних проблема.

2. Главни део часа

Питање. Написати формулу којом се израчунава на колико се начина од $n \geqslant 0$ датих елемената може изабрати k елемената ако поредак њиховог избора није битан.

Одговор. То је број комбинација k-те класе од n елемената који се означава са C_n^k и једнак је $\dfrac{n!}{k!\,(n-k)!}$.

Задатак 1. Написати функцију којом се израчунава број комбинација k-те класе од n елемената коришћењем помоћне функције за рачунање факторијела.

Очекивано решење.

```
# include <iostream>
usingnamespace std;
int factorial(int n)
  {
  int f=1;
  for (int i=2; i<=n; i++) f*=i;
  return f;
  }
int C(int n, int k)
  { return factorial(n)/(factorial(k)*factorial(n-k));}
main()
  {
  int n,k;
  cout <<"Unesi N i K:";
  cin >> n >> k;
  cout << "C(" << n << "," << k << ")=" << C(n,k);
  }
```

Питање. Која је временска сложеност функције којом се израчунава број комбинација k-те класе од елемената?

Одговор. $O(n)$ – јер рачунање $n!$ захтева $n-1$ множења.

Очигледно је да дати алгоритам, иако једноставан и веома брз, има озбиљан недостатак: он ради коректно само за мале вредности n и k. Пробајте да тестирате програм који користи наведену функцију и видећете да већ за C_{13}^5 не даје коректан резултат. То је због тога што величина $n!$ веома брзо расте са увећавањем n, тако да већ $13! = 6\,227\,020\,800$ превазилази максималну вредност која се може регистровати у 32-битном целом броју, а величина $21! = 51\,090\,942\,171\,709\,440\,000$ се не може сместити ни у 64-битном целом броју. Према томе, наведена функција се може користити само за $n \leqslant 12$ при коришћењу 32-битне аритметике или за $n \leqslant 20$ при коришћењу 64-битне аритметике. Чак и када вредност није велика, при израчунавању међурезултата (факторијела) долази до прекорачења. На пример, $C_{30}^{15} = 155\,117\,520$ се може регистровати коришћењем 32-битне аритметике, али међурезултат $30!$ се не може сместити ни у 64-битном целом броју. Према томе, C_{30}^{15} се не може израчунати коришћењем наведеног поступка.

Проблем са прекорачењем може се превазићи коришћењем добро познате рекурентне формуле $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, за $0 < k < n$ и $C_n^0 = C_n^n = 1$.

Задатак 2. Написати рекурзивну функцију којом се по наведеној формули израчунава број комбинација k-те класе од n елемената.

Решење.

```
int C(int n, int k)
  {
  if (k==0 || k==n) return 1;
  return C(n-1,k-1)+C(n-1,k);
  }
```

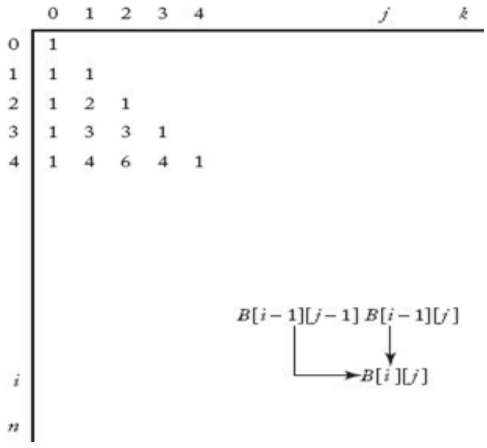
Питање. Да ли се у овом решењу прекорачење појављује као проблем?

Одговор. Не.

Међутим, није тешко приметити да је ово веома лоше решење, јер извршавање траје дуго, због тога што се функција $C(n,k)$ више пута израчунава за исте вредности параметара n и k. На пример, ако се позове функција $C(20,10)$, она ће позвати функције $C(19,9)$ и $C(19,10)$. Функција $C(19,9)$ позива функције $C(18,8)$ и $\boldsymbol{C(18,9)}$, а $C(19,10)$ позива функције $\boldsymbol{C(18,9)}$ и $C(18,10)$. Приметимо да се функција $\boldsymbol{C(18,9)}$ позива два пута. Са повећањем дубине рекурзије број поновљених позива функција брзо расте. На пример, функција $C(17,8)$ се позива три пута (два пута је позива функција $C(18,9)$ и једанпут $C(18,8)$) итд.

Методом динамичког програмирања проблем можемо решити тако да се не троши машинско време на израчунавање већ израчунатих вредности рекурзивне функције. Ради тога, сваку први пут израчунату вредност рекурзивне функције ћемо памтити у низу и – уместо да је, када је поново потребна, израчунавамо – користити запамћену вредност. Зато ћемо креирати дводимензионални низ (матрицу) B у чијем ћемо елементу $B[i][j]$ чувати вредност C_i^j. Аналогно рекурентној формули за израчунавање вредности функције поставићемо формулу за израчунавање вредности елемента дводимензионалног низа

$$B[i][j] = \begin{cases} 1, & j = 0 \text{ или } j = i \\ B[i-1][j-1] + B[i-1][j], & 0 < j < i. \end{cases}$$



Из ове формуле можемо видети да се:

- попуњава само доњи троугао матрице;
- елементи прве колоне матрице ($j = 0$) и дијагонале ($j = i$) лако попуњавају – њихова вредност је 1;
- остали елементи добијају као збир вредности из претходне врсте: дијагонално-лево и непосредно изнад;
- матрица попуњава по врстама.

Задатак 3. Написати функцију која израчунава C_n^k по наведеном алгоритму.

Решење.

```
# include <iostream>
using namespace std;
int C(int n, int k)
  {
  int B[100][100];
  for (int i=0; i<=n; i++)
    for (int j=0;j<=min(i,k); j++)
      if (j==0 || j==i) B[i][j]=1;
      else B[i][j]=B[i-1][j-1]+B[i-1][j];
  return B[n][k];
  }
main()
  {
  int n,k;
  cout <<"Unesi N i K:";
  cin >> n >> k;
  cout << "C(" << n << "," << k << ")=" << C(n,k);
  }
```

Питање. Која је временска сложеност овог алгоритма?

Одговор. $O(n^2)$.

Овакав метод решавања, у коме се један проблем своди на једноставније потпроблеме, који се прво решавају, а њихова решења чувају у низу или некој другој структури података и служе за решавање све сложенијих проблема до полазног, назива се *динамичко програмирање*. Ток решавања проблема овом методом, што је у великој мери илустровано претходним примером, јесте следећи:

1. Формулисати проблем као функцију од неколико аргумената (бројевних, стрингова итд).
2. Свести решење проблема на решење аналогних потпроблема чији параметри, по правилу, имају мање вредности. Пожељно је да се ова веза, ако је то могуће, искаже у виду рекурентне формуле.
3. Задати почетне вредности функције – вредности скупа аргумената за које је функција тривијална.
4. Креирати низ (или другу структуру података) за чување вредности функције. По правилу, ако функција зависи од једног целобројног параметра, користи се једнодимензионални низ, за функције од два целобројна параметра – дводимензионални низ итд.
5. Организовати попуњавање низа прво са почетним вредностима, а затим осталим на основу рекурентне везе.

Задатак 4. Ако се на табли димензија $n \times m$ (n редова и m колона) у горњем левом углу налази шаховски краљ, коме је дозвољено кретање једно поље доле или десно, написати програм којим се одређује број маршрута које га воде до доњег десног угла.

Решење. Ако су врсте табле нумерисане од 0 до $n-1$, а колоне од 0 до $m-1$, почетни положај краља је $(0,0)$, а циљни $(n-1, m-1)$. Нека је $B(a,b)$ број маршрута које воде од почетног поља $(0,0)$ до поља (a,b). Анализирајмо могуће вредности $B(a,b)$:

- ако је $a = 0$, поље се налази у првој врсти и до њега се може стићи на јединствен начин – кретањем удесно. Према томе је $B(0,b) = 1$ за свако b;
- аналогно за $b = 0$ и свако a важи $B(a,0) = 1$;
- број маршрута које воде у поље (a,b) за $a > 0$ и $b > 0$ једнак је збиру броја маршрута које воде до суседног поља слева и суседног поља изнад, тј. $B(a,b) = B(a,b-1) + B(a-1,b)$.

Дакле, важи рекурентна веза:

$$B(a,b) = \begin{cases} 1, & a = 0 \text{ или } b = 0 \\ B(a,b-1) + B(a-1,b), & a > 0 \text{ и } b > 0 \end{cases}$$

коју ћемо искористити за формирање матрице вредности.

Сада можемо написати тражени програм:

```
# include <iostream>
using namespace std;
int BrojMarsruta(int n, int m)
{
  int B[100][100], i, j;
  for (j=0; j<m; j++)
    B[0][j]=1;    // Prva vrsta je popunjena jedinicom.
  for (i=1;i<n; i++)    // i se menja od 1 do n-1.
  {     // Popunjavanje i-te vrste.
    B[i][0]=1;    // Element prve kolone jednak je 1.
    for (j=1; j<m; j++)  // Popunjavanje ostalih elemenata i-te vrste.
      B[i][j]=B[i][j-1]+B[i-1][j];
  }
  return B[i-1][j-1];
}
main()
{
  int n,m;
  cout << "Unesi koordinate ciljnog polja ->";
  cin >> n >> m;
  cout << "Broj marsruta =" << BrojMarsruta(n+1,m+1) << endl;
}
```

Задатак 5. У табели димензија $n \times n$, за $n < 30$, поља су попуњена целим бројевима. Ако су врсте и колоне квадрата означене бројевима од 0 до $n-1$, написати програм којим се исписује маршрута којом се из горњег левог поља $(0,0)$ стиже до доњег десног поља $(n-1, n-1)$ уз поштовање следећих услова:

- са сваког поља је дозвољено прећи само на поље испод или на поље десно од тог поља;
- од свих маршрута, које задовољавају претходни услов, исписати ону чија је сума цифара, у пољима преко којих се иде, максимална.

Ово је проблем оптимизације који, да би се решио методом динамичког програмирања, мора задовољавати *принцип оптималности*, по коме: *оптимално решење проблема мора садржати и оптимално решење сваког свог потпроблема.* У овом проблему то својство је испуњено јер оптимална маршрута, која представља решење проблема, мора бити оптимална на сваком делу пута. Заиста, ако бисмо до неког поља (u, v) преко кога иде оптимална маршрута имали боље решење од оног које је трасирано оптималном маршрутом, цела сума оптималне маршруте би се увећала – што представља контрадикцију тврђењу да је маршрута оптимална.

Ради илустровања проблема узмимо квадратну матрицу A следећег садржаја:

$$\begin{array}{ccccc} 4 & 3 & 5 & 7 & 5 \\ 1 & 9 & 4 & 1 & 3 \\ 2 & 3 & 5 & 1 & 2 \\ 1 & 3 & 1 & 2 & 0 \\ 4 & 6 & 7 & 2 & 1 \end{array}$$

Тражимо путеве, који полазе од поља $(0,0)$, до свих осталих поља матрице, самим тим и до циљног поља $(n-1, n-1)$. Ради тога, креираћемо помоћну квадратну матрицу B, истих димензија као и A, чија ћемо поља попуњавати сумом цифара оптималних путева до њих. У пољу $(0,0)$ матрице B је број $A(0,0)$ јер пут до њега се поклапа са самим пољем. Лако се попуњавају и поља $(0,1)$ и $(1,0)$ матрице B. До њих воде јединствени путеви, па је њихова вредност редом: $A(0,0) + A(0,1)$ и $A(0,0) + A(1,0)$. До поља $(1,1)$ може се стићи из $(0,1)$ или из $(1,0)$. Према томе, у поље $B(1,1)$ треба уписати $\max\{B(0,1) + A(1,1), B(1,0) + A(1,1)\}$. Попуњавање осталих поља матрице B је аналогно. Ако је пут који води до неког поља јединствен, у то поље се уписује сума бројева дуж пута. Таква су поља $(i,0)$ и $(0,i)$. Ако се у поље (i,j) може доћи са поља којима је израчуната вредност, тада је вредност која се уписује у то поље $\max\{B(i,j-1), B(i-1,j)\} + A(i,j)$.

Задатак 5.1. На основу претходне анализе написати рекурентну формулу којом се израчунава вредност поља $B(i,j)$.

Одговор.

$$B(i,j) = \begin{cases} A(0,0), & i=0,\ j=0 \\ A(0,j)+B(0,j-1), & i=0,\ j>0 \\ A(i,0)+B(i-1,0), & j=0,\ i>0 \\ A(i,j)+\max\{B(i-1,j),B(i,j-1)\}, & i>0,\ j>0 \end{cases}$$

Задатак 5.2. Испишите матрицу B и на њој трасирајте оптималну маршруту од поља $(0,0)$ до поља $(n-1,n-1)$.

Одговор.

4	–	7	–	12	–	19	–	24
|		|					|
5		16	–	20	–	21		27
|		|					|
7		19		25		26		29
|		|			|		|
8		22		26		28		29
|		|					
12		28	–	35	–	37	–	38

Задатак 5.3. Напишите програм који користећи установљену рекурентну везу формира матрицу B и исписује оптималну маршруту од поља $(0,0)$ до поља $(n-1,n-1)$.

Решење.

```
# include <iostream>
using namespace std;
int a[13][13],b[13][13];
void pisi(int x, int y)
  {
  if (x>0 && y>0)
    if (b[x-1][y]>b[x][y-1])
    {
    pisi(x-1,y);
    cout << "dole ";
    }
    else
    {
    pisi(x,y-1);
    cout << "desno ";
    }
  else     // x==0 ili y==0
    {
    while (y--) cout << "desno ";
    while (x--) cout << "dole ";
    }
  }
```

```
main()
  {
  int n,i,j;
  cout << "Unesi N -> ";
  cin >> n;
  cout << "\nUnesi vrednosti u poljima matrice:\n";
  for (i=0;i<n;i++)
    for (j=0;j<n;j++)
      cin >> a[i][j];
  b[0][0]=a[0][0];
  for (i=1;i<n;i++)
    {
    b[0][i]=a[0][i]+b[0][i-1];
    b[i][0]=a[i][0]+b[i-1][0];
    }
  for (i=1;i<n;i++)
    for (j=1;j<n;j++)
      b[i][j]=a[i][j] + max(b[i-1][j],b[i][j-1]);
  cout << "Maksimalni zbir je " << b[n-1][n-1] <<
    ", a postize se ovako:" << endl;
  pisi(n-1,n-1);
  }
```

Тестирајте програм за задату матрицу:

```
Unesi N -> 5
Unesi vrednosti u poljima matrice:
4 3 5 7 5
1 9 4 1 3
2 3 5 1 2
1 3 1 2 0
4 6 7 2 1
Maksimalni zbir je 38, a postize se ovako:
  desno dole dole dole dole desno desno desno
```

3. Завршни део часа

Како бисмо увидели предности примене методе динамичког програмирања, направићемо малу компарацију ефикасности решавања проблема – одређивања оптималне маршруте – са и без примене ове методе.

Питање 3.1. Ако се не би користила метода динамичког програмирања, ради одређивања оптималне маршруте би се морале формирати суме по свим маршрутама од горњег левог до доњег десног угла. Колико има таквих маршрута за $n = 9$?

Одговор. Тестирањем решења трећег задатка за $n = 9$ и $m = 9$ добија се да је број маршрута од горњег левог до доњег десног угла 12870.

Питање 3.2. Да ли се тражени број маршрута може добити и израчунавањем броја комбинација k-те класе од n елемената? Образложити одговор.

Одговор. Да. За, на пример $n = 9$, то је C_{16}^{8}. Заиста, ако је матрица 9×9, од горњег левог до доњег десног угла се увек стиже у $2*8$ корака, и то: 8 – десно и 8 – доле. Одатле следи да, ако кретање надесно означимо са 0, а надоле са 1, тражени број је број свих шеснаесторки од 0 и 1 са тачно 8 јединица.

На пример:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	0	1	0	0	1	1	1	0	0	0	0	0
0	1	0	0	1	1	0	1	1	1	1	0	1	0	0	0
1	0	1	1	1	1	1	0	0	0	0	0	0	1	1	0

Одавде је очигледно да је $C_{16}^{8} = 12870$ – број подскупова од позиција на које можемо стављати јединице. Тај број можемо добити применом програма који је решење задатка 2, а који израчунава C_n^k.

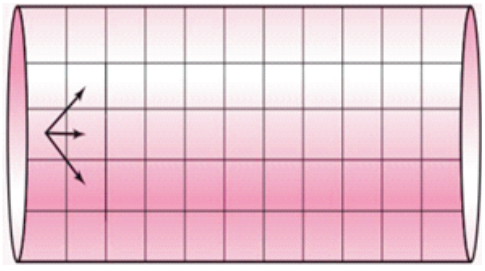
Питање 3.3. За $n = 9$ колико поља табеле, која се користи у току примене методе динамичког програмирања, треба попунити да би се одредила оптимална маршрута?

Одговор. Треба попунити 81 поље табеле димензија 9×9.

Домаћи задатак

1. Бегунац покушава да побегне из замка, који се састоји од $n \times n$ квадратних соба. Неке од соба су затворене и у њих се не може ући. Бегунац се налази у соби у горњем левом углу, а излаз је кроз собу у доњем десном углу. Ако може да се креће само доле и десно, написати програм којим се одређује број путева којима може доћи до излазне собе. Матрицом $A(n \times n)$ дата је мапа проходности соба.

2. Елементи целобројне матрице $A(n \times m)$ су савијени у цилиндар тако да се прва и последња врста матрице додирују. Ако се робот креће са левог краја ка десном уз дозвољена кретања једно поље горе-десно, десно, доле-десно, одредити путању којом се мора кретати тако да збир вредности поља преко којих прелази буде најмањи. Одредити и тај збир.



3. Ако се у банкомату налазе новчанице чија је вредност a_0, a_1, ... , a_{k-1} динара, написати програм којим се кориснику исплаћује износ од N динара са минималним бројем новчаница. Ако је исплата немогућа, даје се одговарајућа порука. Претпоставка је да се у банкомату налази довољно новчаница сваког типа.

Математичка гимназија, Краљице Наталије 37, Београд
E-mail: mcabark@gmail.com

ОБАВЕШТЕЊА

24. БАЛКАНСКА ОЛИМПИЈАДА ИЗ ИНФОРМАТИКЕ

Двадесетчетврта Балканска олимпијада из информатике (BOI) одржана је ове године у Никозији (Кипар). Учествовало је 11 екипа у званичној конкуренцији, као и једна гостујућа. Србију су представљали:

1. *Душан Живановић*, Гимназија „Светозар Марковић“, Ниш,
2. *Ненад Баук*, Математичка гимназија, Београд,
3. *Никола Спасић*, Гимназија „Јован Јовановић Змај“, Нови Сад.
4. *Лука Вукелић*, Математичка гимназија, Београд.

Руководиоци екипе били су *др Драган Урошевић*, Математички институт, Београд и *Демјан Грубић*, Нови Сад.

Наши такмичари су остварили добар резултат. Душан и Ненад су освојили сребрне медаље, а Никола бронзану (Луки је медаља измакла за једно место).

57. МЕЂУНАРОДНА МАТЕМАТИЧКА ОЛИМПИЈАДА

Педесетседма Међународна математичка олимпијада (IMO) одржана је од 6. до 16. јула ове године у Хонг Конгу. Учествовало је 109 екипа. Србију су представљали:

1. *Игор Медведев*, Математичка гимназија, Београд,
2. *Никола Павловић*, Гимназија „Јован Јовановић Змај“, Нови Сад,
3. *Алекса Милојевић*, Математичка гимназија, Београд,
4. *Огњен Тошић*, Математичка гимназија, Београд,
5. *Алекса Константинов*, Математичка гимназија, Београд,
6. *Никола Садовек*, Математичка гимназија, Београд.

Они су освојили једну сребрну медаљу (Алекса Милојевић) и 4 бронзане (Алекса Константинов, Игор Медведев, Огњен Тошић и Никола Садовек), док је Никола Павловић похваљен. Екипно су заузели 40. место, а најбољи су били такмичари из САД.

Руководиоци екипе су били *Душан Ђукић*, Машински факултет и *Марко Радовановић*, Математички факултет, Београд.