Милан Вугделија

ДИНАМИЧКО ПРОГРАМИРАЊЕ

У овом чланку ће бити речи о једном начину решавања неких алгоритамских проблема, у којима се тражи оптимално решење. Погледајмо најпре на примеру у чему се састоји идеја, а затим ћемо кроз дискусију истаћи битне претпоставке за примену овог метода.

Нека је дат квадрат попуњен целим бројевима. Са сваког поља је дозвољено прећи само на поље испод или на поље десно од тог поља. Потребно је изабрати пут од горњег левог поља до доњег десног поља, тако да збир бројева у пољима преко којих се иде, буде максималан.

Једна могућност решавања овог проблема је генерисање свих могућих путева и памћење оног пута који тренутно има највећи збир. Ова идеја није нарочито добра јер број могућих путева расте експоненцијалном брзином са порастом величине квадрата.

Изложићемо сада једну другачију идеју. Ради лакшег изражавања уведимо неколико ознака, и то:

– $A(i,j)$ за број који се налази у пољу (i,j),

– $M(i,j)$ за максимални збир на путу од поља $(1,1)$ до поља (i,j),

– $P(i,j)$ за оптималан пут до поља (i,j) (тј. онај са збиром $M(i,j)$),

– $Q(i,j)$ за проблем налажења пута $P(i,j)$ и збира $M(i,j)$.

Да бисмо сада решили $Q(n,n)$, довољно је да решимо $Q(n,n-1)$ и $Q(n-1,n)$. Оптимални збир $M(n,n)$ се може израчунати као

$$M(n,n) = \max(M(n,n-1), M(n-1,n)) + A(n,n)$$

а тада се $P(n,n)$ добија додавањем поља (n,n) на онај од путева $P(n-1,n)$ или $P(n,n-1)$ који даје већи збир M. Ово стога што оптималан пут до последњег поља садржи у себи оптималан пут до претпоследњег поља, а претпоследње поље може бити само $(n-1,n)$ или $(n,n-1)$.

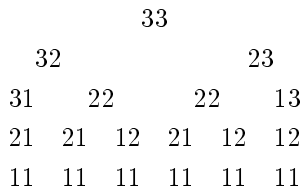
Према томе, решавање полазног проблема се своди на решавање два потпроблема потпуно истог типа али ниже димензије, плус два једноставна корака — поређење два броја и сабирање. Рекурзија овде такође није добро решење јер се већина потпроблема своди на два мања потпроблема, па би укупно било потребно решити експоненцијално много потпроблема уместо само $n * n$, колико их уствари има. Ово стога што се у случају рекурзије многи потпроблеми јављају и решавају више пута. Очигледно би било добро када би сваки потпроблем био решаван само једном. То се може обезбедити тиме што бисмо потпроблеме решавали редом од најједноставнијих према сложенијим и при томе памтили добијена решења. Програм написан по овој идеји изгледао би овако:

```
var a,m:array[1..30,1..30] of integer;
i,j,n:integer;
procedure pisi(a,b:integer);
{var i:integer;}
begin
  if (a>1) and (b>1)
  then
    if m[a-1,b]>m[a,b-1]
      then begin pisi(a-1,b); write('dole ') end
      else begin pisi(a,b-1); write('desno ') end
  else
    if a=1 then for i:=2 to b do write('desno ')
      else for j:=2 to a do write('dole ');
end;
begin {program}
  readln(n);
  for i:=1 to n do
    for j:=1 to n do
      read(a[i,j]);
  m[1,1]:=a[1,1];
  for i:=2 to n do
  begin
    m[1,i]:=a[1,i]+m[1,i-1];
    m[i,1]:=a[i,1]+m[i-1,1]
  end;
  for i:=2 to n do
    for j:=2 to n do
      if m[i,j-1]<m[i-1,j]
        then m[i,j]:=a[i,j] + m[i-1,j]
        else m[i,j]:=a[i,j] + m[i,j-1];
  writeln('Maksimalni zbir je ',m[n,n],' a postiže se ovako:');
  pisi(n,n);
end.{program}
```

Истакнимо особине које је проблем морао имати да би као метод за решавање могло бити изабрано динамичко програмирање.

Прво, оптимално решење проблема садржи у себи оптимална решења својих потпроблема. На пример, нека се на оптималном путу $P(x,y)$ до поља (x,y) прелази преко поља (u,v). Тада је део пута $P(x,y)$ до поља (u,v) оптималан за одговарајући потпроблем $Q(u,v)$, тј. тај део пута $P(x,y)$ је управо $P(u,v)$. Ова чињеница се лако доказује свођењем на контрадикцију: претпостави се да потпроблем $Q(u,v)$ има боље решење и то се доведе у контрадикцију са оптималношћу решења полазног проблема $Q(x,y)$. За овакве проблеме каже се да имају оптималну подструктуру.

Друго, скуп потпроблема је релативно мали, типично полином од улазне величине (овде $n * n$). У исто време, рекурзивним алгоритмом се многи потпроблеми креирају више пута, и сваки пут се изнова решавају. За проблеме са овом особином кажемо да имају преклапајуће потпроблеме. Ефикасност динамичког програмирања долази до изражаја баш на оваквим проблемима, јер се сваки потпроблем креира и решава само једном, а онда се комбинују решења потпроблема да би се решили проблеми вишег реда, све до полазног проблема. Да би се боље уочила ова предност, погледајмо како би изгледало стабло рекурзивних позива у претходном проблему за $n = 3$:



Види се да се већ проблем $Q(2,2)$ (и цело стабло његових потпроблема) јавља два пута, а $Q(1,1)$ чак шест пута. Укупно се креира и решава 19 проблема уместо само 9 колико их има различитих. За веће n губици су још израженији.

Размотримо сада један нови проблем, са намером да илуструјемо како се долази до решења које користи идеје динамичког програмирања. То је проблем налажења најдужег неопадајућег подниза за дати низ.

Подниз неког низа је низ који се добија избацивањем неких (могуће ниједног) елемената полазног низа. Формално, за низ $Z = (z_1, z_2, \ldots, z_k)$ кажемо да је подниз низа $X = (x_1, x_2, \ldots, x_n)$, ако постоји строго растући низ целих бројева (индекса) i_1, i_2, $\ldots$, i_k таквих да за свако j од 1 до k важи $x_{i_j} = z_j$. Проблем се састоји у следећем: за дати низ наћи његов најдужи неопадајући подниз.

Могућност формирања свих поднизова датог низа (са памћењем најдужег нерастућег) није добра јер низ дужине n има 2^n поднизова, па би то био експоненцијални алгоритам. Пошто нема јасне идеје како одредити елементе низа који остају у поднизу, анализирајмо мало природу проблема. Примећујемо да ако је последњи елемент низа X уједно и највећи, онда

$$\mathrm{nnp}(X_n) = (\mathrm{nnp}(X_{n-1}), x_n),$$

где је X_k низ који се састоји од првих k елемената низа X, $\mathrm{nnp}(X_k)$ је најдужи неопадајући подниз низа X_k, а x_k је k-ти елемент низа X. Ако последњи елемент низа није највећи, онда је $\mathrm{nnp}(Xn)$ једнак

$$\mathrm{nnp}(X_{n-1}) \quad \text{или} \quad (\mathrm{nnp}(X_j), x_n)$$

и то оном који је дужи од ова два низа, при чему је X_j најдужи почетни подниз од X, чији nnp завршава елементом не већим од x_n. X_j се може наћи следећим поступком:

```
j=n
ponavljaj
  j=j-1
dok nije (j=0) ili (nnp(Xj) završava brojem ne većim od xn)
```

Из претходног разматрања следи да када се од nnp неког низа X_k одбаци последњи елемент, добија се nnp неког почетног подниза низа X_k. Другим речима, оптимално решење проблема садржи у себи оптимална решења потпроблема, где потпроблемима сматрамо налажење $\mathrm{nnp}(X_k)$, $k = \overline{1,n}$. У овом контексту полазни проблем има оптималну подструктуру.

Приметимо и то да за налажење $\mathrm{nnp}(X_n)$ треба знати $\mathrm{nnp}(X_j)$ за све $j = \overline{k,n-1}$ па проблем има преклапајуће потпроблеме, због чега рекурзија не би била добро решење. Према томе, као идеја се намеће динамичко програмирање.

Потребно је за свако $k = \overline{1,n}$ редом користећи решене потпроблеме и горњу релацију, одредити дужину $\mathrm{nnp}(X_k)$ и његов завршни елемент. Ако постоји више могућих завршних елемената, узети најмањи. На основу дужине и последњег елемента у најдужем неопадајућем поднизу лако се реконструише сам подниз помоћу једне кратке (и брзе) рекурзивне процедуре. Следи програм који решава овај проблем:

```
var
  x:array[1..50] of integer;
  dnnp:array[0..50] of byte; { dnnp[k] je dužina niza nnp(Xk) }
  pe:array[1..50] of byte; { pe[k] je poslednji element niza nnp(Xk) }
  i,j,k,n:byte;
  xkjemax:boolean;
procedure pisi(n:byte);
var m:byte;
begin
  if n>0 then
  begin
    m:=n;
    while dnnp[m]=dnnp[m-1] do m:=m-1;
    pisi(m-1);
    write(' ',pe[n]);
  end
end;
begin
  write('Koliko elemenata ima niz?  ');
  readln(n);
  writeln('Unesite elemente niza redom.  ');
  for i:=1 to n do read(x[i]);
  dnnp[0]:=0; dnnp[1]:=1; pe[1]:=x[1];
  for k:=2 to n do
  begin
    xkjemax:=true;
    for i:=1 to k-1 do if x[k]<x[i] then xkjemax:=false;
    if xkjemax then
      begin
        dnnp[k]:=dnnp[k-1]+1;   pe[k]:=x[k]
```

```
      end
    else
      begin
        j:=k;
        repeat j:=j-1;
        until (j=0) or (pe[j]<x[k]);
        if (dnnp[k-1]<dnnp[j]+1) or
          ( (dnnp[k-1]=dnnp[j]+1) and (x[k]<pe[k-1]) )
        then
          begin
            dnnp[k]:=dnnp[j]+1;   pe[k]:=x[k]
          end
        else
          begin
            dnnp[k]:=dnnp[k-1];   pe[k]:=pe[k-1]
          end
        end {if not xkjemax}
    end; {for k}
    writeln('Najduži neopadajući podniz je dužine ',dnnp[n]:3,'.');
    writeln('Na primer, sledeći neopadajući podniz ima tu dužinu:');
    pisi(n);
    writeln
end.
```

Навешћемо још неколико познатих проблема који се ефикасно решавају динамичким програмирањем. Читалац може вежбати уочавање оптималности подструктуре и преклапања потпроблема, као и саму имплементацију идеје динамичког програмирања.

1. Проблем најдужег заједничког подниза. Дата су два низа X од m и Y од n елемената. Наћи низ највеће могуће дужине који је подниз и за X и за Y.

2. Проблем најкраћег адитивног ланца. Адитивни ланац је низ у коме је сваки елемент осим првог, збир нека два претходна (могуће иста) елемента. За задати цео број n треба наћи најкраћи адитивни ланац који почиње бројем 1 а садржи задати број n.

3. Оптимална триангулација полигона. Нека је дат n-тоугао координатама својих темена и функција f, која пресликава троуглове у реалне бројеве, на пример збир дужина страница троугла. Потребно је дати n-тоугао по одабраним дијагоналама раставити на $n-2$ троугла, тако да збир вредности функције f на та $n-2$ троугла буде максималан.

4. Оптимално заграђивање производа матрица. Дати су бројеви A_0, A_1, $\ldots$, A_n. Ако су матрице M_1, $\ldots$, M_n редом димензија $A_0 \times A_1$, $A_1 \times A_2$, $\ldots$, $A_{n-1} \times A_n$, треба у производу $M_1 \times M_2 \times \cdots \times M_n$ поставити заграде тако да се резултујућа матрица (која је увек иста без обзира на распоред заграда) добије са што је могуће мање множења бројева.