Милан Чабаркапа

BACKTRACKING АЛГОРИТМИ

Backtracking алгоритме карактерише трагање за решењем не по задатим правилима рачунања већ методом покушаја и грешке. Ови проблеми се често најприродније изражавају рекурзивно и захтевају испитивање коначног броја потпроблема. Задаци који се решавају бектрекингом, по правилу, припадају једној од три класе:

- наћи бар једно решење проблема;
- наћи сва могућа решења;
- по задатом критеријуму одредити оптимално решење.

Један од класичних проблема: распоређивање осам краљица на шаховској плочи тако да се међусобно не нападају – једноставно се решава бектрекинг техником.

Општи облик *backtracking* алгоритма може се представити на следећи начин:

1. За дати проблем треба одредити низ решења $(x[1], x[2], \ldots, x[n])$ који задовољава скуп услова и ограничења. (У проблему 8 краљица треба одредити низ $x[1]$, $x[2]$, ... , $x[8]$, који репрезентују положаје краљица у врстама шаховске плоче од 1 до 8, тако да се никоје две краљице међусобно не нападају.)

2. За сваки елемент $x[k]$ траженог низа $x[1..n]$ постоји њему припадајући скуп $a[k]$ из кога бирамо кандидата за k-ту координату делимичног решења. (У примеру осам краљица низ x има осам елемената $x[1]$, $x[2]$, ... , $x[8]$ који узимају вредности на скуповима $a[1]$, $a[2]$, ... , $a[8]$, који изражавају положај краљице у врсти. Овде су сви скупови низа a једнаки, тј. $a[1] = a[2] = \cdots = a[8] = \{1..8\}$.)

3. Елементима низа $x[1]$, ... , $x[n]$ вредности се додељују обично слева надесно. Претпоставимо да је нађено првих $k-1$ елемената низа $X = (x[1], x[2], \ldots, x[k-1], ?, \ldots, ?)$; тада избор елемента низа $x[k]$ зависи од неких ограничења и услова. (У проблему 8 краљица се проверава: да ли би дошло до узајамног нападања краљица постављених у врстама са положајима $x[1]$, ... , $x[k-1]$ са краљицом која би се у k-тој врсти поставила на положај $x[k]$, тј. да ли постоји краљица у некој врсти i $(< k)$ и положајем $x[i]$ која се напада са краљицом у врсти k и положајем $x[k]$ – нападају се ако се налазе у истој колони $x[i] == x[k]$ или на истој дијагонали $\mathrm{abs}(i-k) == \mathrm{abs}(x[i]-x[k])$.) Ако су услови и ограничења задовољени за неку вредност коју $x[k]$ узима из скупа $a[k]$, то не значи да се $(x[1], .., x[k])$ може проширити до потпуног решења, али треба покушати. Зато се прелази на избор $x[k+1]$, затим $x[k+2]$ итд.

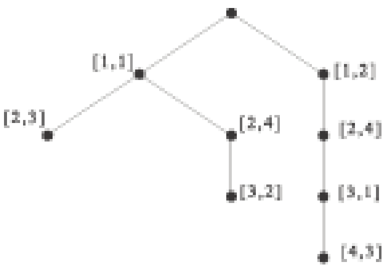
4. Ако ни један избор $x[k]$ не доводи до потпуног решења, онда је неопходно вратити се на претходни корак, пробати са новом вредношћу за $x[k-1]$ у делимичном решењу $(x[1], \ldots, x[k-1], ?, \ldots, ?)$, и поново пробати са избором елемента $x[k]$, проверавајући да ли се $(x[1], \ldots, x[k], ?, \ldots, ?)$ може развити до потпуног решења $(x[1], \ldots, x[n])$. Повратак назад може ићи чак до првог елемента.

Следећа слика илуструје примену backtracking алгоритма за распоређивање 4 краљице на шаховској плочи 4×4. Симбол # означава поља која угрожава краљица из прве врсте, @ – поља која угрожава краљица из друге врсте, & – поља која угрожава краљица из треће врсте.

1	#	#	#
#	#	2	@
#	@	#	@
#		@	#

1	#	#	#
#	#	@	2
#	3	#	@
#	@	&	#

#	1	#	#
#	#	#	2
3	#	@	#
&	#	4	@



Претрага се може графички илустровати стаблом. Корен стабла (0-ти ниво) представља празан вектор решења. Његови „синови“ су кандидати за избор $x[1]$, и у општем случају, чворови k-тог нивоа су кандидати за избор $x[k]$, под условом да су изабрани претци $x[1]$, $x[2]$, ..., $x[k-1]$.

Рекурзивна шема реализације backtracking алгоритма се може представити на следећи начин:

```
void Backtrack(vektor, k)
  {
  if (⟨vektor je resenje⟩) ⟨ispisati resenje⟩
  else
    {
    ⟨formirati skup Ak - kandidata za k-tu komponentu⟩
    for (x∈Ak) Backtrack(vektor || x,k+1)
    // znak || oznacava prosirenje vektora resenja komponentom x
    }
  }
```

Овај алгоритам се може модификовати тако да се провера – да ли се избором x дошло до решења – реализује у `for`-циклусу, уместо непосредно при уласку у функцију:

```
void Backtrack(vektor, k)
  {
  ⟨formirati skup Ak - kandidata za k-tu komponentu⟩
  for (x∈Ak)
    {
```

```
    if (⟨vektor⟩ || x - je resenje⟩)
      ⟨ispisati resenje⟩
    else Backtrack(vektor || x,k+1)
    // znak || oznacava prosirenje vektora resenja komponentom x
    }
  }
```

Ова шема backtracking алгоритма је примењена у следећем решењу проблема 8 краљица. У њему се функција за распоређивање `postavi()` позива из `main()` функције, у којој се индикатор успешности претраживања `nadjeno` иницијализује са 0 (неуспех). У функцији `postavi()` ако се пронађе решење, он добија вредност 1 (успех). Низ $x[\,]$ се индексира од 1 до 8.

```
// Rasporedjivanje 8 kraljica tako da se medjusobno ne napadaju
# include "math.h"
# define N 8
int x[9],nadjeno;
void postavi(int k)
  {
  void pisi(int n);
  int napadaju_se(int k);
  for(int i=1;i<=8 && !nadjeno;i++)
    {
    x[k]=i;  //izbor x[k] za koje se proverava da li je
           //(x[1],...,x[k]) delimicno resenje
    if (!napadaju_se(k))  //ako je (x[1],...,x[k]) delimicno resenje
      if (k==N)  // ono je i potpuno kada je k jednako N(=8)
        {
        pisi(N);  // ispis resenja
        nadjeno=1;  // indikator uspeha postavlja na 1
        }
      else postavi(k+1);  // pokusaj da se od delimicnog resenja
           //(x[1],...,x[k]) dobije potpuno;
           //ako je uspesno, globalna promenljiva nadjeno
           // postaje 1 i prekida se dalje pretrazivanje;
           //ako je promenljiva nadjeno i dalje jednaka 0
           //opoziva se tekuci izbor x[k]
           //(u ovom slucaju ne mora sa x[k]=0),
           // vec se proba sa novom vrednoscu za x[k]
           // iz skupa dopustenih vrednosti
    }
  }
// provera da li se kraljice postavljene na pozicijama
//(x[1],...,x[k-1]) napadaju sa kraljicom postavljenom na x[k]
int napadaju_se(int k)
  {
  for(int i=1;i<=k-1;i++)
    if (x[i]==x[k] || abs(i-k)==abs(x[i]-x[k])) return 1;
  return 0;
  }
void pisi(int n)
```

```
    {
    printf("Resenje:  n");
    for(int i=1;i<= n;i++)
      {
      for(int j=1;j<= n;j++)
        if (x[i]==j)
          printf("K ");
        else printf("+ ");
      printf(" n");
      }
    }
  void main()
    {
    nadjeno=0;  // indikator koji postaje 1 kada se nadje resenje
    postavi(1);  // poziv funkcije za rasporedjivanje osam kraljica
    }
```

Ако треба одредити сва решења, тада се после нађеног решења не прекида даље трагање, па се зато индикатор `nadjeno` из претходног решења не користи. Овај проблем решава модификована функција `postavi()`:

```
void postavi(int k)
  {
  for(int i=1;i<=8;i++)
    {
    x[k]=i;
    if (!napadaju_se(k))
      if (k==N)
        pisi(N);
      else postavi(k+1);
    }
  }
```

Проблем се може решити и тако да се уместо провере циклусом да ли краљица у врсти k на i-тој позицији напада краљице постављене у врстама $1, \ldots, k-1$ – уведу три низа:

1. $v[1..8]$ – контролише да ли су угрожена поља вертикала 1..8 шаховске плоче.

2. $g[0..14]$ – контролише угроженост главне дијагонале и њених паралела. Приметимо да је разлика индекса врсте и индекса колоне за сва поља: главне дијагонале једнака 0, прве паралеле испод главне дијагонале $1, \ldots$, седме паралеле испод дијагонале 7, прве паралеле изнад -1, седме паралеле изнад -7. Зато је најприродније да се низ g индексира од $-7..7$. Међутим, низови у **C**-у се могу индексирати само од 0, па се транслацијом за 7 уводи коресподенција за контролу угрожености поља главне дијагонале и њених паралела индексима 0..14. Низ $g[0..14]$ се индексира са: 0 – за седму паралелу изнад главне дијагонале, $\ldots$, 7 – за главну дијагоналу, $\ldots$, и 14 – за седму паралелу испод главне дијагонале. У складу са овим низ $g[\,]$ се индексира од 0.

3. $s[2..16]$ – контролише угроженост споредне дијагонале и њених паралела. Како је збир индекса врсте и колоне за сва поља: споредне дијагонале једнак

9, ..., седме паралеле изнад споредне дијагонале једнак 2, ..., седме паралеле испод споредне дијагонале једнак 16, то се низ $s[\,]$ ради контроле споредне дијагонале и њених паралела индексира од 2 до 16.

Применом *backtracking* технике проблем 8-краљица решава следећи програм, у коме се елементи низова $v[\,]$, $g[\,]$ и $s[\,]$ иницијализују јединицом – да би репрезентовали слободне вертикале, дијагонале и њихове паралеле. У програму се коментарима објашњавају поједини делови кода.

```
#include ⟨iostream.h⟩
// v[1..8] - kontrolise ugrozenost polja vertikale:  1 - slobodna, 0 - ne
int v[9]=0,1,1,1,1,1,1,1,1;  // element v[0] se ne koristi
// g[0..14] - kontrolise ugrozenost glavne dijagonale i njenih paralela
int g[15]=1,1,1,1,1,1,1,1,1,1,1,1,1,1,1; //
//s[2..16] - kontrolise ugrozenost sporedne dijagonale i njenih paralela
int s[17]=0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
int x[9]; // polozaj kraljice u vrstama od 1 ..  8
int res=0; // redni broj resenja
void Ispis()
  {
  res++;
  cout ⟨⟨ "Resenje " ⟨⟨ res ⟨⟨ endl;
  for (int i=1; i<=8; i++)
    {
    for (int j=1; j<=8; j++)
      if (x[i]==j)
        cout ⟨⟨ " *";
      else cout ⟨⟨ " -";
    cout ⟨⟨ endl;
    }
  cout ⟨⟨ "****************" ⟨⟨ endl;
  }
// da li se kraljica moze postaviti u k-toj vrsti na poziciji i
int MozeNa(int k,int i)
  {
  return v[i] && g[k-i+7] && s[k+i];
  }
void postavi(int k)
  {
  for (int i=1;i<=8;i++)
    if (MozeNa(k,i)) // da li moze u k-toj vrsti na i-toj poziciji
      {
      x[k]=i;  // u k-toj vrsti postavlja kraljicu na i-tu poziciju
      v[i]=0;  // markira i-tu kolonu kao ugrozenu
      g[k-i+7]=0;  // markira paralelu glavne dijagonale kao ugrozenu
      s[k+i]=0;  // markira paralelu sporedne dijagonale kao ugrozenu
      if (k==8)
        Ispis();  // ispis - postavljeno je 8 kraljica
      else postavi(k+1);  // inace, postavi kraljicu u k+1-voj vrsti
      // da bi se probalo sa kraljicom na drugoj poziciji k-te vrste,
      //uklanja se sa pozicije i,
      // a njome ugrozena polja se markiraju kao slobodna
```

```
        v[i]=1;
        g[k-i+7]=1;
        s[k+i]=1;
        }
    }
void main()
    {
    postavi(1);
    }
```

Пример. „**Ранац**“. За n различитих типова предмета су познати тежине $t_1, \ldots, t_n$ и цене $c_1, \ldots, c_n$. Одредити које предмете треба ставити у ранац, тако да њихова укупна тежина није већа од s, а да је цена максимална, ако није ограничен број предмета неког типа који се може ставити у ранац.

На пример, за ранац капацитета 10, и 3 типа предмета тежине: 2, 4, 5, и цене: 2, 5, 5, најбољи избор је један предмет типа 1 и два предмета типа 2.

Ако означимо количину предмета типа i са x_i, треба максимизирати

$$c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n x_n,$$

уз ограничење

$$t_1 \cdot x_1 + t_2 \cdot x_2 + \cdots + t_n \cdot x_n \leqslant s,$$

где је $0 \leqslant x_i \leqslant [s/t_i]$ – квадратне заграде означавају цео део.

```
# include ⟨iostream.h⟩
int n;    // broj predmeta
int s;    // kapacitet ranca
int t[20];    // tezine predmeta
int c[20];    // cena predmeta
int x[20];    // evidentira kolicinu izabranih predmeta
int MaxX[20];    // evidentira najbolji izbor predmeta
int MaxCena; // maksimalna cena
// izbor kolicine ucesca k-tog predmeta u rancu
void Dodaj(int k,int s, int TCena)
// k - indeks elementa niza x[k] koji evidentira kolicinu k-tog predmeta
// s - preostali kapacitet u rancu
// TCena - cena predmeta tekuceg izbora
  {
  if (k>n)  // nadjeno resenje
    {
    if (TCena>MaxCena)  // zapamtiti ga ako je bolje
      {
      for (int j=1; j<=n;j++) MaxX[j]=x[j];
      MaxCena=TCena;
      }
    }
  else
    for (int i=0;i<=s/t[k];i++)
      { // k-ti predmet se moze uzeti najvise u kolicini s/t[k]
      x[k]=i; // izbor k-tog predmeta u kolicini i
      Dodaj(k+1,s-i*t[k],TCena+i*c[k]);
```

```
        }
    }
void main()
  {
  int i;
  cout ⟨⟨ " nUnesite broj predmeta --⟩"; cin ⟩⟩ n;
  cout ⟨⟨ "Koliki teret se moze staviti u ranac --⟩ "; cin ⟩⟩ s;
  cout ⟨⟨ "Unesite za svaki predmet tezinu i cenu:  n";
  for (i=1; i<=n; i++)
    {
    cout ⟨⟨ "t[" ⟨⟨ i ⟨⟨ "]="; cin ⟩⟩ t[i];
    cout ⟨⟨ "c[" ⟨⟨ i ⟨⟨ "]="; cin ⟩⟩ c[i];
    }
  MaxCena=0;
  Dodaj(1,s,0);
  cout ⟨⟨ "Maksimalna cena --⟩" ⟨⟨ MaxCena ⟨⟨ endl;
  cout ⟨⟨ "Za sledeci izbor predmeta --⟩ " ⟨⟨ endl;
  for (i=1;i<=n;i++)
    cout ⟨⟨ "Predmet " ⟨⟨ i ⟨⟨ ".  u kolicini " ⟨⟨ MaxX[i] ⟨⟨ endl;
  }
```

Решење се може учинити ефикаснијим ако се пре позива функције `Dodaj()` изврши сортирање по вредности предмета (критеријум количник цене и тежине).

Задаци за вежбу

1. Дато је n тегова тежина $t[0..n-1]$. Одредити бар једну поделу на две групе тако да је разлика сума тежина тегова у тим групама најмања.

2. Од n предмета са тежинама $a[0..n-1]$ и ценама $c[0..n-1]$ издвојити оне чија је укупна тежина већа од 30 kg, а цена најмања. Решити ако:

а) понављање предмета је дозвољено;

б) сваки тип предмета се може изабрати тачно једном.

3. S динара треба раситнити помоћу новчаница вредности $v[0..n-1]$. Сваке врсте новчаница има довољно много. Наћи сва решења.

4*. Написати програм којим се у скупу природних бројева одређују решења једначине $x_1^4 + x_2^4 + \cdots + x_{15}^4 = 2003$, уз услов $x_1 \geqslant x_2 \geqslant \ldots \geqslant x_{15}$.

5. Дато је n коцки нумерисаних од 1 до n, чије су дужине ивица дате низом $d[1..n]$, а боје знаковним низом $b[1..n]$. Написати програм који:

а) исписује све куле од m коцки;

б) одређује кулу са највећим бројем коцки (од којих је изграђена),

ако сваке две суседне коцке имају различиту боју и ако им је дужина ивица у нерастућем поретку.

6. Дат је низ целих бројева дужине N и цео број R. Написати програм који у изразу:

$$(\ldots((a[1]?a[2])?a[3])\ldots)?a[n]$$

поставља знаке $+$, $-$, $*$ и $/$ тако да је вредност израза једнака R. Наћи сва решења.

7. Исписати индексе оних елемената низа чији је збир једнак датом броју n.

8. Агенција за посредовање знајући висине n младожења и n удавача врши упаривање тако да разлика висина код сваког пара није већа од d. Одредити што је могуће више парова који задовољавају дати критеријум.

9. На шаховској табли димензија 8×8 постављена је краљица у врсти m и колони n. Распоредити ако је могуће још 7 краљица тако да се свих 8 краљица међусобно не напада.

10. На шаховској табли димензија 8×8 у првих m врста је постављено m краљица које се међусобно не нападају. Распоредити ако је могуће још $8 - m$ краљица у врстама од $m + 1$ до 8 тако да се свих осам краљица међусобно не напада.

11. Ако је дата шаховска табла димензија $n \times n$ и стартна позиција скакача $X0$,$Y0$, написати програм који одређује једну путању од $n^2 - 1$ скокова којим скакач обилази плочу посећујући свако поље тачно једанпут.

12. Ако је дата шаховска табла димензија $n \times n$ и стартна позиција скакача $X0, Y0$, написати функцију која одређује све путање од $n^2 - 1$ покрета којим скакач обилази таблу посећујући свако поље тачно једанпут.

13. Нека је дато n концентричних кругова таквих да сваки од њих има m отворених врата. Проласком кроз било која врата додељује се известан број ненегативних поена. Ако је дата матрица A димензије $n \times m$ чији елемент $a[i][j]$ означава број поена који се осваја проласком кроз j-та врата i-тог круга (кроз сваки круг се пролази тачно једанпут), написати програм који трасира пут кроз n врата тако да се сакупи дати број поена s.

14. Дат је лавиринт у коме су нуле проходна, а јединице непроходна поља. Два поља су суседна ако су им заједничке странице. Написати програм којим се одређују сви путеви од датог поља (StX, StY) до датог поља $(EndX, EndY)$.

15. Дат је лавиринт у коме су нуле проходна, а јединице непроходна поља. Два поља су суседна ако су им заједничке странице. Написати програм којим се одређује најкраћи пут од датог поља (StX, StY) до датог поља $(EndX, EndY)$.