Material de Referência

Programação Dinâmica

1. Números Fibonacci varsão iterativa (usando Programação Dinâmica):

```
int fibonacci(int n )

{

int last=1, nextToLast=1, answer=1;

 if( n <= 1 )

           return 1;

 for( int i = 2; i<=n; i++ )

 {

           answer = last + nextToLast;

           nextToLast = last;

           last = answer;

} return answer;

}
```

2. Encontrar o maior e o menor elemento de um vetor de inteiros A[1..n], n ≥ 1:

```
void MaxMin4 (int Linf, int Lsup, int *Max,
int *Min)

 {

 int Max1, Max2, Min1, Min2, Meio;

 if (Lsup - Linf <= 1)

 {

           if (A[Linf-1] < A[Lsup-1])

           {

           *Max = A[Lsup-1]; *Min = A[Linf-
           1];

            }

           else { *Max = A[Linf-1]; *Min =
           A[Lsup-1]; }

}

 else {

 Meio = (Linf+Lsup)/2;

MaxMin4(Linf, Meio, &Max1, &Min1);
MaxMin4(Meio+1, Lsup, &Max2, &Min2);

 if (Max1 > Max2) *Max = Max1; else *Max
= Max2; if (Min1 < Min2) *Min = Min1;
else *Min = Min2; }

}
```

3. Encrontar o menor valor em n Matrizes:

```
#define Maxn 10

 int main(int argc, char *argv[])

 {

           int i, j, k, h, n, temp;

           int b[Maxn+1];

           int m[Maxn][Maxn];

           printf(“Numero de matrizes n: ”);
           scanf(“%d”, &n);

            getchar();

           printf(“Dimensoes das matrizes:
           ”);

 for (i = 0; i <= n; i++)

scanf(“%d”, &b[i]);

for (i = 0; i < n; i++)

           m[i][i] = 0;

for (h = 1; <= n-1; h++) {

           for (i = 1; i <= n-h; i++) {

            j = i+h; m[i-1][j-1] = INT_MAX;

           for (k = i; k <= j-1; k++) {

 temp = m[i-1][k-1] + m[k][j-1] + b[i-1] *
b[k] * b[j];

 if (temp < m[i-1][j-1])

 m[i-1][j-1] = temp; }

 printf(“m[%d][%d] = %d\n”, i-1, j-1, m[i-1]
[j-1]);

 } putchar(’\n’);

 }

 return 0; }
```

Programação Dinâmica - Algoritmo de Kadane

O problema de segmento de soma máxima é a tarefa de encontrar um subvetor com a maior soma possível. Por exemplo, para a seguinte sequência de valores -2,1,-3,4,-1,2,1,-5,4; o segmento de soma máxima é 4,-1,2,1 com soma 6. O algoritmo trivial seria esse:

```
void seg_max(int *v, int n, int & x, int &y , int & max){
int i,j;
int soma;
max = -1;
for(i=0;i<n;i++){
soma = 0;
for(j=i;j<n;j++){
soma += v[j];
if( soma > max ){
max = soma;
x = i;
y = j;
}
}
}
}

int main(){
int x,y,max;
int v[] = {-2,1,-3,4,-1,2,1,-5,4};
seg_max(v,9,x,y,max);
printf("segmento de soma maxima [%d-%d] com soma %d\n", x,y,max);
}
```

Saída
segmento de soma maxima [3-6] com soma 6

Programação Dinâmica - Soma de Subconjunto (Subset Sum)

Soma de Subconjunto

Problema : Dado um conjunto de n números a[i] que a soma total é igual a M, e para algum K≤M, se existe um subconjunto dos números tais que a soma desse subconjunto dá exatamente K?

Solução 1:
Vamos usar uma tabela unidimensional m[0..M], m[b] indica se b pode ser alcançado.

```
for(i=0;i<=M;i++) m[i] = 0;
m[0]=1;
for(i=0;i
for(j=M; j>= a[i]; j--)
m[j] |= m[j-a[i]]
```

Observações: A idéia original é usar uma matriz bidimensional m[0..i-1] [0..M], onde cada linha depende apenas da linha anterior. Mas utilizando um compressão de estados, podemos utilizar apenas um vetor unidimensional.
Mas precisamos escrever o loop j na ordem reversa para evitar confusões.

Variações

Existem várias variações do problema da soma de subconjuntos.

Doces para duas crianças: Os valores dos a[i]’s representam os valores dos doces. Nós queremos dividir igualmente os doces entre as duas crianças ou da maneira mais justa. Agora o problema não é encontrar um subconjunto
com soma fixa K. Nós queremos encontrar um certo K mais próximo de M/2.
http://br.spoj.pl/problems/TESOURO/http://www.spoj.pl/problems/FCANDY/

Troco (coin change): Agora cada a[i] representam moedas, você quer passar um troco de exatamente K. Talvez existam várias maneiras de você fazer isso, então você quer minimizar (ou maximiza) o número de moedas que você precisa usar.
A estrutura da solução não precisa ser alterada, nós precisamos apenas mudar o significado do vetor m. Agora m[b] não será 0 ou 1, será

exatamente o número mínimo de moedas que você precisa para alcançar b.
https://br.spoj.pl/problems/MOEDAS/http://www.spoj.pl/problems/NOCHANGE

Contando o troco (couting change): Os a[i]s continuam representando moedas, agora você quer saber quantas maneiras você pode passar o troco K. O número de maneiras de trocar uma quantidade A usando N tipos de moedas é igual a:
1. O número de maneiras de trocar a quantidade A usando todas as moedas anteriores, +
2. O número de maneiras de trocar a quantidade A-D usando todos os N tipos de moedas, onde D é o valor da n enésima moeda.

A árvore de recursão do processo vai reduzir gradualmente o valor de A, então podemos usar estas regras, para determinar o número de maneiras de trocar um certo valor
1. Se A == 0, então vamos contar 1 maneiras.
2. Se A <=0, então vamos contar 0 maneiras
3. Se N tipos de moedas == 0, então vamos contar 0 maneiras.

```
int coin[] = {1,5,10,25,50};
int count[MAX_MONEY+1];
memset(count,0,sizeof(count));
count[0] = 1;
for (int i=0;i
for (int j=coin[i];j<=MAX_MONEY;++j)
count[j] += count[j-coin[i]];
```

Programação Dinâmica - Maior Subsequência Comum

O problema da maior subsequência comum (longest commom subsequence) (LCS) é o problema de encontrar a maior subsequência comum a duas sequências X e Y. Este é um problema clássico para compação de arquivos e bioinformática.

Uma subsequência de uma sequência X de caracteres pode ser obtida removendo alguns caracteres dessa sequência. Por exemplo,

Considere a seguinte sequência X = ABCBDAB, podemos obter 2^|X| subsequência distintas de X. Por exemplo, BCBDAB e ACDAB são uma subsequência de X.

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#define MAX 100
using namespace std;
int main()
{

char X [MAX];
char temp[MAX];
int m;

scanf("%s",X);

m = strlen(X);

int i,j,index;

for(i = 0; i < 1<<m ; i++){
index = 0;
for(j=0; j<m; j++){
if( (i & (1<<j)) != 0 ){
temp[index++] = X[j];
}
}
temp[index] = '\0';
printf("%s\n",temp);
}

return 0;
}
```

Entrada
BCBD
Saída
B
C
BC
B
BB
CB

BCB
D
BD
CD
BCD
BD
BBD
CBD
BCBD

Sub estrutura Ótima

Caso Base:

Se i=0 ou j=0 então c[i,j] = 0. Se umas das sequências é nula então a maior sequência comum também é nula.

Se X[i]=X[j] então c[i,j] = c[i-1,j-1] + 1. Se o último caracter for igual nas duas sequências então este caractere está na maior subsequencia comum e podemos reduzir o problema para c[i-1,j-1].

Se X[i]!=X[j] então c[i,j] = max(c[i,j-1],c[i-1,j]). Se o último caracter não for igual nas duas sequências então temos duas opções:
deletamos o último caractere da sequência Y e comparamos com a sequência X completa(c[i,j-1])
deletamos o último caractere da sequência X e comparamos com a sequência Y completa(c[i-1,j])

A solução ótima será dada pelo máximo entre os dois subproblemas.

```
int n,m;
int i,j;

scanf("%s",X);
scanf("%s",Y);

n = strlen(X);
m = strlen(Y);

for(i=0;i<=n;i++)
c[i][0] = 0;

for(j=0;j<=m;j++)
c[0][j] = 0;


for(i=1;i<=n;i++)
for(j=1;j<=m;j++)
if(X[i-1]==Y[j-1])
c[i][j] = c[i-1][j-1] + 1;
else
c[i][j] = max( c[i-1][j] , c[i][j-1]);



printf("%d\n",c[n][m]);
```

Entrada
ABCBDAB BDCABA
Saída
4

A **maior subsequência comum** pode ser extraída de uma tabela da seguinte maneira:

```
i = n;
j = m;

stack <char> pilha;

while(i!=0 && j!=0){
if(X[i-1]==Y[j-1]){
pilha.push(X[i-1]);
i--;
j--;
}else if(c[i-1][j] > c[i][j-1]){
i--;
}else{
j--;
}
}

while( !pilha.empty() ){
cout << pilha.top() ;
pilha.pop();
}
cout << endl;
```

A **maior subsequência comum** pode ser obtida recursivamente da seguinte maneira:

```
void lcs(int i,int j){
if(i!=0 && j!=0){
if(X[i-1]==Y[j-1]){
lcs(i-1,j-1);
printf("%c",X[i-1]);
}else if(c[i-1][j] > c[i][j-1]){
lcs(i-1,j);
}else{
lcs(i,j-1);
}
}
}
```

Chamada

```
lcs(n,m);
cout << endl;
```

Para calcular a solução de um subproblema em particular, precisamos apenas de duas linhas para obter a solução dos subproblemas c[i-1,j], c[i,j-1], c[i-1,j-1].

Podemos otimizar a utilização da memória aproveitando este fato da seguinte maneira:

```
char X[MAXN];
char Y[MAXN];
int c[2][MAXN];

int n,m;
int i,j;

scanf("%s",X);
scanf("%s",Y);

n = strlen(X);
m = strlen(Y);

for(j=0;j<=m;j++)
c[0][j] = 0;

for(i=1;i<=n;i++)
for(j=1;j<=m;j++)
if(X[i-1]==Y[j-1])
c[i&1][j] = c[(i-1)&1][j-1] + 1;
else
c[i&1][j] = max(c[(i-1)&1][j] , c[i&1][j-1]);

printf("%d\n",c[n&1][m]);
```

Este problema tem variações interessantes:

Encontrar a **menor supersequência comum entre duas string**. Por exemplo, a menor supersequência comum entre AAATTT e GAATCT é GAAATCTT. Observe que AAATTT e GAATCT são subsequência de GAAATCTT e GAAATCTT é a menor sequência
com essa propriedade.Esse problema foi explorado em:
http://br.spoj.pl/problems/PARQUE/

Encontrar **a menor distância entre duas strings A e B**. A distância entre duas string é dada pelo menor número de operações necessária para transformar A em B. As operações permitidas são deletar um caractere, adicionar um caractere ou substituir um caractere por outro. Pode ser encontrado aqui:
http://www.spoj.pl/problems/EDIST/

Links interessantes:

http://cs.nyu.edu/~yap/classes/funAlgo/05f/lect/l7.pdfhttp://comscigate.com/Books/contests/icpc.pd