**算法分析第3次作业**

算法分析题

3-1

S[i]表示从0到i的最长单调递增子序列

当i==0时，S[i]=={a[0]}；

当i>0时，S[i]=max{S[k](0<=k<i&&a[k]<=a[i])}+a[i];max表示满足条件的最长单调递增子序列。

伪代码如下

GetMaxLenSubsequence(int n)

{

S[0]={a[0]};

for(i=1;i<n;i++)

{

for(j=0,temp={};j<i;j++)

{

if(a[j]<=a[i]&&len(temp)<S[j])

{

temp=S[j];

}

}

S[i]=temp+a[i];//‘+’表示序列中添加a[i]

}

return S[n-1];

}

使用了两重循环，时间复杂度为O(n^2)。

3-3

设m(i,j,k）表示背包容量为j，体积为k时，可选择物品为i，i+1，...，n时的最优值。可建立递归式。

当j>=wi&&k>=bi，m(i,j,k)=max{m(i+1,j,k),m(i+1,j-wi,k-bi)+vi}，m(n,j,k)=vi；

当0<=j<wi||0<=k<bi，m(i,j,k)=m(i+1,j,k)，m(i,j,k)=0；

伪代码如下

Template<class Type>

void Knapsack(Type c,int w[],int b[],int c,int d,Type \*\*\*m)

{

int jMax=min(w[n]-1,c),kMax=min(b[n]-1,d);

for(int j=0;j<=jMax;j++)

{

for(int k=0;k<=d;k++)

{

m[n][j][k]=0;

}

}

for(int k=0;k<=kMax;k++)

{

for(int j=0;j<=c;j++)

{

m[n][j][k]=0;

}

}

for(int j=w[n];j<=c;j++)

{

for(int k=b[n];k<=d;k++）

{

m[i][j][k]=v[n];

}

}

for(int i=n-1;i>1;i--)

{

jMax=min(w[i]-1,c);

kMax=min(b[i]-1,d);

for(int j=0;j<=jMax;j++)

{

for(int k=0;k<=c;k++)

{

m[i][j[k]=m[i+1][j][k];

}

}

for(int k=0;k<=kMax;k++)

{

for(int j=0;j<=d;j++)

{

m[i][j][k]=m[i+1][j][k];

}

}

for(int j=w[i];j<=c;j++)

{

for(int k=b[i];k<=d;k++)

{

m[i][j][k]=max(m[i+1][j][k],m[i+1][j-w[i]][k-b[i]]+v[i]);

}

}

}

m[1][c][d]=m[2][c][d];

if(c>=w[1]&&d>=b[1])

{

m[1][c][d]=max(m[2][c][d],m[2][c-w[1]][d-b[1]]+v[1]);

}

}

m[1][c][d]为最优值。遍历三维数组，时间复杂度为O(ncd)。

算法实现题

3-3

设m(i,j)表示从第i个石子到第j个石子的最小或最大得分，sum(i)表示从第一个石子到第i个石子的得分之和。

由于得分=此前得分+当前得分，当前得分为一固定值，即石子的分数之和，所以需取得此前得分的最值。

当求最小得分时m(i,j)=min{m(i,k)+m(k+1,j))+sum(j)-sum(i-1)}(i<=k<j)；

同理，当求最大得分时m(i,j)=max{m(i,k)+m(k+1,j))+sum(j)-sum(i-1)}(i<=k<j)；

m(1,n)为最终值。

伪代码如下：

Void msum(int a[],int n)

{

sum[1]=a[1];

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

{

sum[i]=sum[i-1]+a[i];

}

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

{

for(int i=1;i<=n-len+1;i++)

{

int j=i+len-1;

m[i][j]=INF;

for(int k=i;k<j;k++)

{

m[i][j]=min/max(m[i][j],m[i][k]+m[k+1][j]+sum[j]-sum[i-1]);

}

}

}

}

三重循环，外面两重循环遍历斜对角矩阵，时间复杂度O(n^2)，内循环每次枚举j-i=len-1次，计算次数为O(n)，总时间复杂度为O(n^3)。

3-13

设S(i,j)表示从i位到第j位的数，m(i,j)表示S(0,i)的最大j乘积，可以写出递归式

m(i,j)=max{m(t,j-1)\*S(t+1,i)}(0<=t<i-1)；

m(i,1)=S(0,i)

m(n-1,k)为最终解

伪代码如下

void multiply(int n,int k)

{

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

{

m[i][1]=S(0,i);

}

for(int j=2;j<=k;j++)

{

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

{

for(int t=0;t<i-1;t++)

{

m[i][j]=max(m[t][j-1]\*S(t+1,i));

}

}

}

}

三重循环，最外层枚举k-1次，内层每层计算O(n)，时间复杂度O(k\*n^2)

3-14

设K(i)表示第i个编号的商品的购买总数，P(i)表示第i个编号的商品的正常单价，A(i),B(i),C(i),D(i),E(i)分别表示第i种优惠方案中购买a、b、c、d、e商品的数量，f(i)表示第i种优惠方案的优惠价，m（a,b,c,d,e）表示购买商品组合(a,b,c,d,e)的最少费用，则当单独购买时，m(a,b,c,d,e)=∑K(i)\*P(i)，当购买第i种优惠方案时m(a,b,c,d,e)=m(a-A[i],b-B[i],c-C[i],d-D[i],e-E[i])+f(i)(a>=0,b>=0,c>=0,d>=0,e>=0)，通过优惠方案取代单独购买，最终得到最小值。

伪代码如下

void mincost(int K[],int P[],int A[],int B[],int C[],int D[],int E[],int f[])

{

for(int a=0;a<=K[0],a++)

{

for(int b=0;b<=K[1];b++)

{

for(int c=0;c<=K[2];c++)

{

for(int d=0;d<=K[3];d++)

{

for(int e=0;e<=K[3];e++)

{

k[0]=a;

k[1]=b;

k[2]=c;

k[3]=d;

k[4]=e;

int prize=0;

for(int i=0;i<n1;i++)

{

prize+=k[i]\*P[i];

}

for(int i=0;i<n2;i++)

{

aa=k[0]-A[i];

bb=k[1]-B[i];

cc=k[2]-C[i];

dd=k[3]-D[i];

ee=k[4]-E[i];

if(aa>=0&&bb>=0&&cc>=0&&d>=0&&ee>=0&&m(aa,bb,cc,dd,ee)+f(i)<prize)

{

prize=m(aa,bb,cc,dd,ee)+f(i);

}

}

m(a,b,c,d,e)=prize;

}

}

}

}

}

}

最终结果为m(K[0],K[1],K[2],K[3],K[4])。

使用了n1重外循环，内有n1和n2的遍历，时间复杂度O(k^n1\*max(n1,n2))。

3-17

设dist(i,j)表示A[i-1]和B[j-1]的距离，m(i,j)表示子串A(0~i)和B(0~j)的扩展距离，则有递归式

m(i,j)=min{m(i-1,j)+k,m(i,j-1)+k,m(i-1,j-1)+dist(i,j)}

m(0,0)=0

m(n1.n2)为最终解

伪代码如下

void extensiveDistance(string A,string B,int k)

{

int n1=A.length,n2=B.length;

for(int i=0;i<=n1;i++)

{

for(int j=0;j<=n2;j++)

{

if(i+j>0)

{

m[i][j]=min(m[i-1][j]+k,m[i][j-1]+k);

}

if(i\*j>0)

{

m[i][j]=min(min[i][j],m[i-1][j-1]+dist(i,j));

}

}

}

}

使用两重循环，时间复杂度为O(n1\*n2)