3-4

等价于背包问题，具有最优子结构性质

对于子问题：

$$\max \sum_{k=i}^{n} c_k x_k$$

$$\sum_{k=i}^{n} a_k x_k \le j$$

其最优值为 $m(i,j)$

可得

$$m(i,j) = \begin{cases} \max\{m(i-1,j),\ m(i,j-a_i)+c_i\}, & j \ge a_i \\ m(i-1,j) & 0 \le j < a_i \end{cases}$$

$m(0,j) = m(i,0) = 0$，$m(i,j) = -\infty$，$j<0$.

计算此式得出，$m(n,b)$ 为最优解.

时间复杂度为 $O(nb)$

3-5

问题等价于：

给定 $c>0$，$d>0$，$w_i>0$，$b_i>0$，$v_i>0$，$1 \le i \le n$

求向量 $(x_1, x_2 \dots x_n)$，$x_i \in \{0,1\}$，$1 \le i \le n$

使 $\sum_{i=1}^{n} w_i x_i \le c$，$\sum_{i=1}^{n} b_i x_i \le d$ 且有 $\max \sum_{i=1}^{n} v_i x_i$

具有最优子结构性质

其子问题：$\max \sum_{t=i}^{n} v_t x_t$

$$\begin{cases} \sum_{t=i}^{n} w_t x_t \le j \\ \sum_{t=i}^{n} b_t x_t \le k \\ x_t \in \{0,1\},\ i \le t \le n \end{cases}$$

最优值为 $m(i,j,k)$

计算 $m(i,j,k)$ 递归式如下：

$$m(i,j,k)=\begin{cases}\max\{m(i+1,j,k),\ m(i+1,j-w_i,k-b_i)+v_i\} & j\ge w_i \text{ 且 } k\ge b_i\\ m(i+1,j,k), & 0\le j<w_i \text{ 或 } 0\le k<b_i\end{cases}$$

$$m(n,j,k)=\begin{cases}v_n & j\ge w_n \text{ 且 } k\ge b_n\\ 0 & 0\le j<w_n \text{ 或 } 0\le k<b_n\end{cases}$$

由此求出 $m(n,c,d)$ 最优值，

所用时间为 $O(ncd)$

3-6

递归算法：

```
int Ackermann(int m, int n)
{
    if(m==0) return n+1;
    if(n==0) return Ackermann(m-1, 1);
    else return Ackermann(m-1, Ackermann(m,n-1))
}
```

使用两个数组 val[0:m] 和 ind[0:m]

已计算 A[i, ind[i]] 值存储于 val[i] 中

当计算到 Ackermann(m,n) 时算法结束

```
int  A(int m, int n)
{
    int i.val[ ], ind[ ];
    if (m==0)  return n+1;
    val = new  int[m+1];
    ind = new  int [m+1];
    for(i=0; i<=m; i++)
    {
        val[i]=-1;
        ind[i]=-2;}
    val[0]=1;
    ind[0] =0;
    while(ind[m]<n)
    {val[0]++;
     ind[0]++;
    for(i=0; i<m; i++)
    {    if(ind[i]==1 && ind[i+1]<0)
         {val[i+1]=val[0];
          ind[i+1]=0;}
         if(val[i+1]) == ind[i]
         {val[i+1]=val[0];
          ind[i+1]++;}      }
    }
    return val[m]
}
```

3-4　考虑下面的整数线性规划问题：

$$\max \sum_{i=1}^{n} c_i x_i$$

$$\sum_{i=1}^{n} a_i x_i \leqslant b$$

x_i 为非负整数，$1 \leqslant i \leqslant n$

试设计一个解此问题的动态规划算法，并分析算法的计算复杂性。

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

long long unboundedKnapsack(int n, long long b, vector<long long>& a, vector<long long>&
c) {
    vector<long long> dp(b + 1, 0);

    for (long long j = 1; j <= b; j++) {
        for (int i = 0; i < n; i++) {
            if (j >= a[i]) {
                dp[j] = max(dp[j], dp[j - a[i]] + c[i]);
            }
        }
    }

    return dp[b];
}

int main() {
    int n;
    long long b;

    cout << "Enter the number of items (n): ";
    cin >> n;
    cout << "Enter the capacity (b): ";
    cin >> b;

    vector<long long> a(n), c(n);
    cout << "Enter the weights (a_i) for each item:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    cout << "Enter the values (c_i) for each item:" << endl;
    for (int i = 0; i < n; i++) {
```

```
        cin >> c[i];
    }

    long long maxValue = unboundedKnapsack(n, b, a, c);

    cout << "Maximum value achievable: " << maxValue << endl;

    return 0;
}
```

3-5 给定 n 种物品和一背包。物品 i 的重量是 w_i，体积是 b_i，其价值为 v_i，背包的容量为 C，容积为 D。问：应该如何选择装入背包中的物品，使得装入背包中物品的总价值最大？在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分的物品 i。试设计一个解此问题的动态规划算法，并分析算法的计算复杂性。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> weight = { 600, 400, 200, 200, 300 };
vector<int> volume = { 800, 400, 200, 200, 300 };
vector<int> value = { 8, 10, 4, 5, 5 };

void knapsack_2d(const int &c, const int &d) {
    const int n = value.size();
    //分配内存空间，存储m(i,j,k)
    int ***m = new int **[n + 1];
    for (int i = 0; i <= n; i++) {
        m[i] = new int *[c + 1];
        for (int j = 0; j <= c; j++)
            m[i][j] = new int[d + 1];
    }
    //分配内存空间，存储0-1向量
    int *x = new int[n + 1];

    //初始化m(n,j,k)
    for (int j = 0; j <= c; j++) {
        for (int k = 0; k <= d; k++) {
            m[n][j][k] = 0;
            if (j >= weight[n - 1] && k >= volume[n - 1])
                m[n][j][k] = value[n - 1];
        }
```

```
    }

    //求解子问题
    for (int i = n - 1; i > 0; i--) {
        for (int j = 0; j <= c; j++) {
            for (int k = 0; k <= d; k++) {
                m[i][j][k] = m[i + 1][j][k];
                if (j >= weight[i - 1] && k >= volume[i - 1])
                    m[i][j][k] = max(m[i + 1][j][k], m[i + 1][j - weight[i - 1]][k -
volume[i - 1]] + value[i - 1]);
            }
        }
    }

    //构造最优解
    int temp1 = c, temp2 = d;
    for (int i = 1; i < n; i++) {
        if (m[i][temp1][temp2] == m[i + 1][temp1][temp2])
            x[i] = 0;
        else {
            x[i] = 1;
            temp1 -= weight[i - 1];
            temp2 -= volume[i - 1];
        }
    }
    x[n] = (m[n][c][d] > 0) ? 1 : 0;

    //输出
    cout << "最大价值：" << m[1][c][d] << endl;
    cout << "0-1向量：";
    for (int i = 1; i <= n; i++)
        cout << x[i] << ' ';

    //释放内存空间
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= c; j++)
            delete[] m[i][j];
        delete[] m[i];
    }
    delete[] m;
    delete[] x;
}

//测试程序
```

```
int main(void) {
    int c = 1000;
    int d = 1000;
    knapsack_2d(c, d);

    return 0;
}
```

3-6 Ackerman 函数 $A(m,n)$ 可递归地定义如下：

$$A(m,n)=\begin{cases} n+1 & m=0 \\ A(m-1,1) & m>0,n=0 \\ A(m-1,A(m,n-1)) & m>0,n>0 \end{cases}$$

试设计一个计算 $A(m,n)$ 的动态规划算法，该算法只占用 $O(m)$ 空间。（提示：用两个数组 val[0:m]和 ind[0:m]，使得对任何 i 有 val[i]=$A(i$,ind[i])）。

```
#include <iostream>

int Ackerman(const int& m, const int& n) {
    if (m < 0 || n < 0)
        return -1;
    if (m == 0)
        return n + 1;
    int* val = new int[m + 1];
    int* ind = new int[m + 1];

    val[0] = 1;
    ind[0] = 0;
    for (int i = 1; i <= m; i++) {
        val[i] = -1;
        ind[i] = -2;
    }
    while (ind[m] < n) {
        val[0]++;
        ind[0]++;
        for (int i = 0; i < m; i++) {
            if (ind[i] == 1 && ind[i + 1] < 0) {
                val[i + 1] = val[0];
                ind[i + 1] = 0;
            }
            if (val[i + 1] == ind[i]) {
                val[i + 1] = val[0];
                ind[i + 1]++;
            }
        }
    }
```

```
    int ans = val[m];
    delete[] val;
    delete[] ind;
    return ans;
}

//测试程序
int main(void) {
    std::cout << Ackerman(3, 10);
    return 0;
}
```