1、求解下面的递归式，并针对每个递推式给出一个 Θ 界限。

（1）$T(n)=7T(n/7)+n$

解：$T(n)=7T(n/7)+n$，其中 $a=7, b=7, f(n)=n$。因此，$n^{\log_b a}=n^{\log_1 1}=n=\Theta(n)$，由于 $f(n)=n=\Theta(n)$，应用主定理第二条定理有，$T(n)=\Theta\left(n^{\log_b a}\log n\right)=\Theta(n\log n)$。

（2）$T(n)=8T(n/6)+n^{3/2}\log n$

解：$T(n)=8T(n/6)+n^{3/2}\log n$，其中 $a=8, b=6, f(n)=n^{3/2}\log n$。因此，$n^{\log_b a}=n^{\log_6 8}=\mathrm{O}\left(n^{0.778}\right)$，由于 $f(n)=n^{3/2}\log n=\Omega\left(n^{\log_6 8+\varepsilon}\right)$，其中 $\varepsilon\approx 0.8$，应用主定理情况 3，当 n 足够大时，对于 $c=8/6=4/3$，有：

$af(n/b)=8f(n/6)\log(n/6)\le(4/3)n^{3/2}\log n=cf(n)$，条件成立。因此，应用情况 3 有，$T(n)=\Theta\left(n^{3/2}\log n\right)$。

（3）$T(n)=2T\left(n^{1/3}\right)+1$

解：做代换 $n=2^m$，只考虑 $n^{1/3}$ 为整数的情形。可以得到，$T\left(2^m\right)=2T\left(2^{m/3}\right)+1$，将此式改写为：$S(m)=2S(m/3)+1$。

①对此式应用主定理情况 1，可以得到 $S(m)=\Theta\left(m^{\log_b a}\right)=\Theta\left(m^{\log_3 2}\right)$，所以：

$T(n)=T\left(2^m\right)=S(m)=\mathrm{O}\left(m^{\log_3 2}\right)=\mathrm{O}\left(\left(\log_2 n\right)^{\log_3 2}\right)$

② $S(m)=2S(m/3)+1=1+2\left(1+2S\left(m/3^2\right)\right)=\ldots=1+2+2^2+\ldots+2^k S\left(m/3^k\right)$，当 $m/3^k=1$ 时，有，$k=\log_3 m$，所以 $S(m)=1+2+\ldots+2^{\log_3 m}S(1)=2^{\log_3 m+1}-1$，因此，

$T(n)=T\left(2^m\right)=S(m)=\mathrm{O}\left(2^{\log_3 m}\right)=\mathrm{O}\left(m^{\log_3 2}\right)=\mathrm{O}\left(\left(\log_2 n\right)^{\log_3 2}\right)$

2、给出一个求解背包问题的算法。并说明其正确性。

输入：给定 n 个物品，物品价值分别为 $P_1, P_2, \ldots, P_n$ 物品重量分别 $W_1, W_2, \ldots, W_n$，背包容量为 M。每种物品可部分装入到背包中。

输出：$X_1, X_2, \ldots, X_n, 0 \le X_i \le 1$，使得 $\sum_{1\le i\le n} P_i X_i$ 最大，并且满足 $\sum_{1\le i\le n} W_i X_i \le M$。

解：这个问题的特点是：每种物品只有一件，可以选择放或者不放。利用动态规划思想，子问题为：假设 $f(i,j)$ 表示剩余容量为 j，剩余物品为 $i, i+1, \ldots, n$ 时的最优解的值。DP 求解过程可以这样理解：

对于前 i 件物品，背包剩余容量为 j 时，所取得的最大价值只依赖于两个状态。

状态 1：只要不选第 i 个物品，前 $i-1$ 件物品，背包剩余容量为 j。在该状态下，

$$f[i-1][j]$$。

状态 2：选第 i 个物品，前 $i-1$ 件物品，背包剩余容量为 $j-W[i]$。在该状态下，

$$f[i-1][j-W[i]]+P[i]$$

因为，这里要求最大价值，所以只要从状态 1 和状态 2 中选择最大价值较大的一个即可。

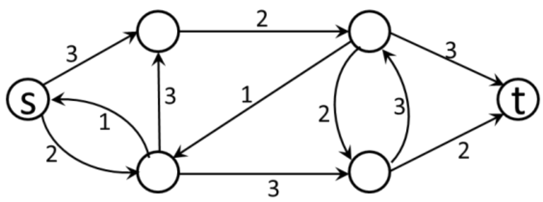
（1）采用二维数组

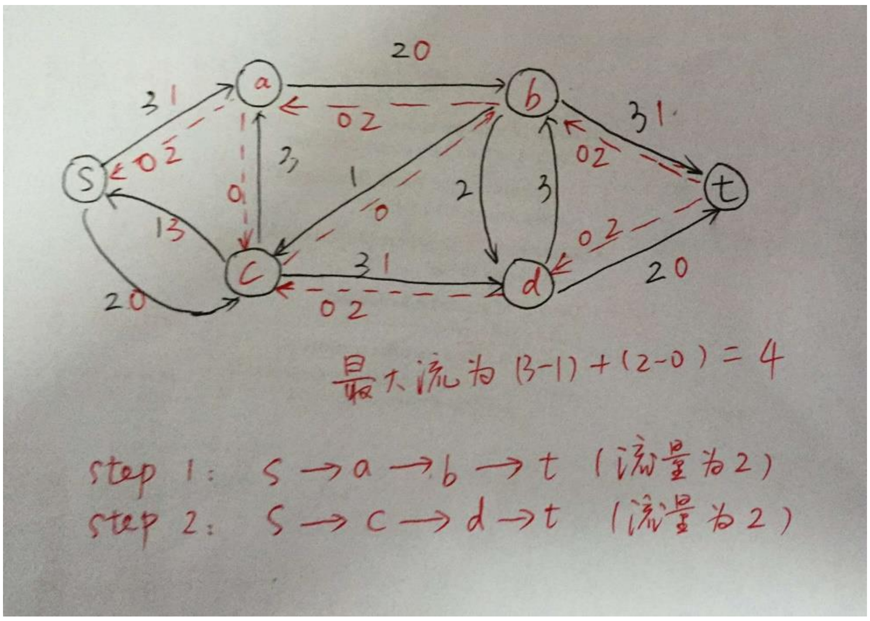
```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    static const int M = 10;                    // 定义空间最大值
    static const int n = 5;                     // 物品个数
    vector<vector<int> > dp(n + 1, vector<int>(M + 1, 0));
    vector<int> W(n + 1), P(n + 1);

    for (int i = 1; i <= n; ++i)
        cin >> W[i] >> P[i];                    // 输入物品i的体积和价值

    for (int i = 1; i <= n; ++i)                // 对每一个物品进行选择
        for (int j = P[i]; j <= M; ++j)         // 对每一个状态进行比较
                                                // 按照状态转移方程进行比较
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - W[i]] + P[i]);

    cout << dp[M] << endl;                      // 输出最后结果

    return 0;
}
```

（2）采用一维数组

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

int main()
{
    static const int M = 10;                    // 定义空间最大值
    static const int n = 5;                     // 物品个数
    vector<int> dp(M + 1, 0);                   // 动态规划数列赋初始值
    vector<int> W(n + 1), P(n + 1);

    for (int i = 1; i <= n; ++i)
        cin >> W[i] >> P[i];                    // 输入物品i的体积和价值

    for (int i = 1; i <= n; ++i)                // 对每一个物品进行选择
        for (int j = M; j >= P[i]; --j)         // 对每一个状态进行比较
                                                // 按照状态转移方程进行比较
            dp[j] = max(dp[j], dp[j - W[i]] + P[i]);

    cout << dp[M] << endl;                      // 输出最后结果

    return 0;
}

```

3、下图是一个流网络，请说明 s 到 t 的最大流值是多少？并标示一种流方式。





4、设 $X[1..n]$，$Y[1..n]$ 为两个数组，每个都包含 n 个有序元素。请设计一个 $O(\log n)$ 时间的算法来找出数组 X 和 Y 中所有 $2n$ 个元素的中位数。

算法： 分别对 X 和 Y 进行二分搜索，对于搜索中值 $middle$，比较 $X[middle]$ 和 $Y[middle]$ 的大小，如果 $X[middle]<Y[middle]$，那么就舍弃 X 的前半部分和 Y 的后半部分。这样下一次搜索的区间就减少了一半。$X[middle]=Y[middle]$ 或者 X 和 Y 的搜索区间长度为 1。这时取 $(X[middle]+Y[middle])/2$ 为返回的中位数。

解：

```cpp
#include<iostream>
using namespace std;

static const int n = 10;
double X[n] = { 1,2,3,4,5,6,7,8,9,10 };
double Y[n] = { 8,9,10,11,12,13,14,15,16,17 };

double middle(int iX, int jX, int iY, int jY)
{
    int midX = (iX + jX) / 2;
    int midY = (iY + jY) / 2;

    if (iX == jX || iY == jY || X[midX] == Y[midY])
        return (X[iX] + Y[iY]) / 2;  // 终止条件
    else
    {
        if (X[midX] < Y[midY])
            // 奇偶讨论，如果搜索范围内有奇数个数，那么右侧左移一次
            //          如果搜索范围内有偶数个数，那么右侧在原位不动
            return middle(midX + 1, jX, iY, midY - ((iY + jY) % 2 ? 0 : 1));
        else
            return middle(iX, midX - ((iX + jX) % 2 ? 0 : 1), midY + 1, jY);
    }
}

int main()
{
    cout << middle(0, 9, 0, 9);
    return 0;
}
```

5、给出 N 个 1-9 的数字（v_1, v_2, …, v_N），不改变它们的相对位置，在中间加入 K 个乘号和 N-K-1 个加号，（括号随便加）使最终结果尽量大。因为乘号和加号一共就是 N-1 个了，所以恰好每两个相邻数字之间都有一个符号。

分析：假设有序列 $a_1, a_2, a_3, ..., a_n$，其目标结果最大值设为 $f(n,c)$，其中 n 表示前 $1 \sim n$ 个数，c 表示其中有 c 个乘号。于是有子问题：序列 $a_1, ..., a_i$，其目标结果最大值设为 $f(i,c)$。

现考察 $f(n,c)$，对乘号个数进行动态规划。其状态取决于：假设将前 $1 \sim n$ 个数在位置 k 处插入乘号。那么 $1 \sim k$ 个数中乘号个数为 $c-1$，而且 $k+1 \sim n$ 个数中全为加号（因为此时乘号已经用完了），于是此时的 $f(n,c) = f(k,c-1) \times \sum_{i=k+1}^{n} a_i$。

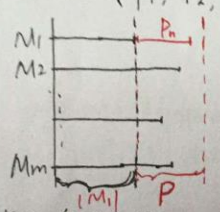
于是可以得到状态转移方程：$f(n,c) = \max\limits_{c \le k < n}\left(f(k,c-1) \times \sum_{i=k+1}^{n} a_i \right)$

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

int main()
{
    int n, k;
    while (cin >> n >> k)
    {
        vector<int> a(n + 1);
        vector<vector<int> > sum(n + 1, vector<int>(n + 1, 0));
        vector<vector<int> > dp(n + 1, vector<int>(k + 1, 0));
        for (int i = 1; i <= n; i++) cin >> a[i];  // 输入待求数组
        for (int i = 1; i <= n; i++)
        {
            for (int j = i; j <= n; j++)
            {
                int s = 0;
                for (int k = i; k <= j; k++)
                    s += a[k];     // sum[i][j]初始化为待求数组i到j的和
                sum[i][j] = s;
            }
        }
        for (int i = 1; i <= n; i++)
            dp[i][0] = sum[1][i];// dp[i][0]为1到i中，乘号个数为0时，数组的最大结果
        for (int j = 1; j <= k; j++)
        {                        // 不断的添加乘号，动态规划求最大值
            for (int i = j + 1; i <= n; i++)
            {                    // 添加乘号时，起始计算位是j+1
                dp[i][j] = -1;      // 开始动态规划
                for (int k = j; k < i; k++)
                    dp[i][j] = max(dp[i][j], dp[k][j - 1] * sum[k + 1][i]);
                                 // 自底向上根据状态转移方程求最大值
            }
        }
        cout << dp[n][k];
    }
    return 0;
}
```

6、给定 n 个任务以及 m 个机器，每个任务所需的处理时间为 $p_1, p_2, \ldots, p_n$。一个任务可由任一台机器执行，但不能拆开分配给多台机器执行。一台机器在一时间只能处理一个任务。有一算法，它将这 n 个任务依次分配给这 m 个机器。算法在分配一任务时，总是将任务分配给到目前为止分配了最少工作时间的机器。请给出该算法的近似比，并证明。

解：

6. $(p_1, p_2, \cdots, p_n)$



$P = \max\{p_1, p_2, \cdots p_n\}$.

$P \le OPT$

$OPT \ge \frac{\sum_{i=1}^{n} p_i}{m}$

$\therefore OPT \ge \max\{P, \frac{\sum_{i=1}^{n} p_i}{m}\}$

$\therefore 2OPT \ge P + \frac{\sum_{i=1}^{n} p_i}{m}$

考虑已分配了 $n-1$ 个任务，并记 M_1 为此时最少分配的机器，则

$|M_1| \le \frac{\sum_{i=1}^{n} p_i}{m}$，且在其他机器 M_1 右侧的长度不超过 P.（若超过，则 M_1 不是 $n-1$ 个任务时长度最少的那个）．故添加 p_n 后，有

$$ALG \le |M_1| + P \le \frac{\sum_{i=1}^{n} p_i}{m} + P \le 2OPT$$

∴近似比为 2.