

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， $n^{\log_b a} = n^{\log_7 1} = n = \Theta(n)$ ，

由于 $f(n) = n = \Theta(n)$ ，应用主定理第二条定理有，

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n)。$$

$$(2) T(n) = 8T(n/6) + n^{3/2} \log n$$

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，

$n^{\log_b a} = n^{\log_6 8} = O(n^{0.778})$ ，由于 $f(n) = n^{3/2} \log n = \Omega(n^{\log_6 8 + \varepsilon})$ ，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：

$af(n/b) = 8f(n/6) \log(n/6) \leq (4/3)n^{3/2} \log n = cf(n)$ ，条件成立。因此，应用情

况 3 有， $T(n) = \Theta(n^{3/2} \log n)$ 。

$$(3) T(n) = 2T(n^{1/3}) + 1$$

解：做代换 $n = 2^m$ ，只考虑 $n^{1/3}$ 为整数的情形。可以得到， $T(2^m) = 2T(2^{m/3}) + 1$ ，

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 $S(m) = \Theta(m^{\log_b a}) = \Theta(m^{\log_3 2})$ ，所以：

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

② $S(m) = 2S(m/3) + 1 = 1 + 2(1 + 2S(m/3^2)) = \dots = 1 + 2 + 2^2 + \dots + 2^k S(m/3^k)$ ，当

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

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

2、给出一个求解背包问题的算法。并说明其正确性。

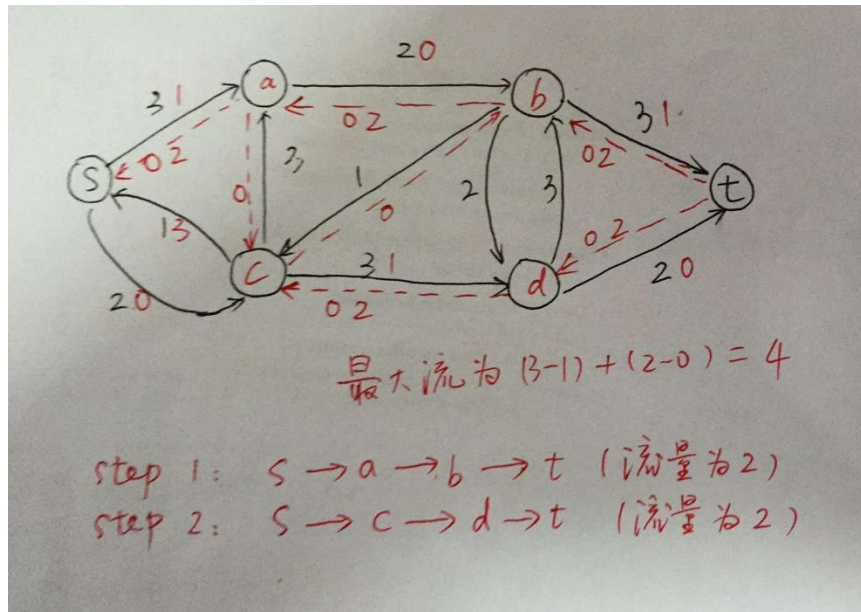
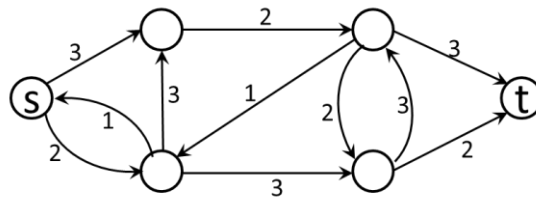
输入：给定 n 个物品，物品价值分别为 P_1, P_2, \dots, P_n 物品重量分别 W_1, W_2, \dots, W_n ，背包容量为 M 。每种物品可部分装入到背包中。

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

解： 使用贪心策略，首先放入单位价值最高的物品，然后依次放入。因此，对于每个物品，首先求出其单位价值，然后按照单位价值降序排列。然后从单位价值最高的物品放入，如果这个物品质量小于背包剩下的容量，那么可以将其全部放入，总价值加上这个物品的价值。否则只能部分放入背包。部分放入背包的价值即为这个物品的单位价值乘以背包剩余容量。这样，最终所得的总价值即为解。

```
01 #include <iostream>
02 #include <algorithm>
03 #include <vector>
04 using namespace std;
05
06 struct Thing
07 {
08     double p;          // 物品价值
09     double w;          // 物品重量
10     double per;        // 物品单位价值
11 };
12
13 int main()
14 {
15     int n, M; cin >> n >> M;          // 输入物品个数和背包容量
16     vector<Thing> list(n);             // 物品信息列表
17     for (int i = 0; i < n; ++i)
18     {
19         cin >> list[i].p >> list[i].w;
20         list[i].per = list[i].p / list[i].w;
21     }
22
23     sort(list.begin(), list.end(), [](Thing a, Thing b) {
24         return a.per < b.per;
25     }); // 按照per值降序排列
26
27     double sum = 0;
28     for (int i = 0; i < n; ++i)          // 对于每个物品
29     {
30         if (M > list[i].w)               // 如果其重量小于背包剩余容量
31         {
32             M -= list[i].w;              // 那么将其完整放入背包
33             sum += list[i].p;             // 总价值加上物品价值
34         }
35         else                             // 否则，将物品部分放入背包
36         {
37             sum += list[i].per * M;       // 可放入价值即为剩余容量乘以单位价值
38             break;                       // 总价值加上部分物品价值
39         }
40     }
41     cout << sum << endl;
42
43     return 0;
44 }
```

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：

```
01 #include<iostream>
02 using namespace std;
03
04 static const int n = 10;
05 double X[n] = { 1,2,3,4,5,6,7,8,9,10 };
06 double Y[n] = { 8,9,10,11,12,13,14,15,16,17 };
07
08 double middle(int iX, int jX, int iY, int jY)
09 {
10     int midX = (iX + jX) / 2;
11     int midY = (iY + jY) / 2;
12
13     if (iX == jX || iY == jY || X[midX] == Y[midY])
14         return (X[iX] + Y[iY]) / 2; // 终止条件
15     else
16     {
17         if (X[midX] < Y[midY])
18             // 奇偶讨论，如果搜索范围内有奇数个数，那么右侧左移一次
19             // 如果搜索范围内有偶数个数，那么右侧在原位不动
20             return middle(midX + 1, jX, iY, midY - ((iY + jY) % 2 ? 0 : 1));
21         else
22             return middle(iX, midX - ((iX + jX) % 2 ? 0 : 1), midY + 1, jY);
23     }
24 }
25
26 int main()
27 {
28     cout << middle(0, 9, 0, 9);
29     return 0;
30 }
31
```

5、给出 N 个 1-9 的数字 (v_1, v_2, \dots, 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 $a_1, a_2, a_3, \dots, a_n$ ，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, \dots, 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：
$$f(n, c) = \max_{c \leq k < n} \left(f(k, c-1) \times \sum_{i=k+1}^n a_i \right)$$

```

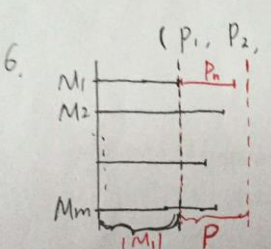
01 #include<iostream>
02 #include<algorithm>
03 using namespace std;
04
05 int main()
06 {
07     int n, k;
08     while (cin >> n >> k)
09     {
10         vector<int> a(n + 1);
11         vector<vector<int>> > sum(n + 1, vector<int>(n + 1, 0));
12         vector<vector<int>> > dp(n + 1, vector<int>(k + 1, 0));
13         for (int i = 1; i <= n; i++) cin >> a[i]; // 输入待求数组
14         for (int i = 1; i <= n; i++)
15         {
16             for (int j = i; j <= n; j++)
17             {
18                 int s = 0;
19                 for (int k = i; k <= j; k++)
20                     s += a[k]; // sum[i][j]初始化为待求数组i到j的和
21                 sum[i][j] = s;
22             }
23         }
24         for (int i = 1; i <= n; i++)
25             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
26         for (int j = 1; j <= k; j++)
27         { // 不断的添加乘号，动态规划求最大值
28             for (int i = j + 1; i <= n; i++)
29             { // 添加乘号时，起始计算位是j+1
30                 dp[i][j] = -1; // 开始动态规划
31                 for (int k = j; k < i; k++)
32                     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
33             }
34         }
35         cout << dp[n][k];
36     }
37     return 0;
38 }
39
40

```

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, \dots, 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，它把这 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, \dots, p_n) $p = \max\{p_1, p_2, \dots, p_n\}$



$$p \leq \text{OPT}$$

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

$$\therefore \text{OPT} \geq \max\left\{p, \frac{\sum_{i=1}^n p_i}{m}\right\}$$

$$\therefore 2\text{OPT} \geq p + \frac{\sum_{i=1}^n p_i}{m}$$

考虑已分配了 $n-1$ 个任务，并记 M_1 为此时最少分配机器，则

$$|M_1| \leq \frac{\sum_{i=1}^{n-1} p_i}{m}$$

且在 M_1 右侧的长度不超过 p 。（若超过，
其他机器
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

$$\text{ALG} \leq |M_1| + p \leq \frac{\sum_{i=1}^n p_i}{m} + p \leq 2\text{OPT}$$

\therefore 近似比为 2.