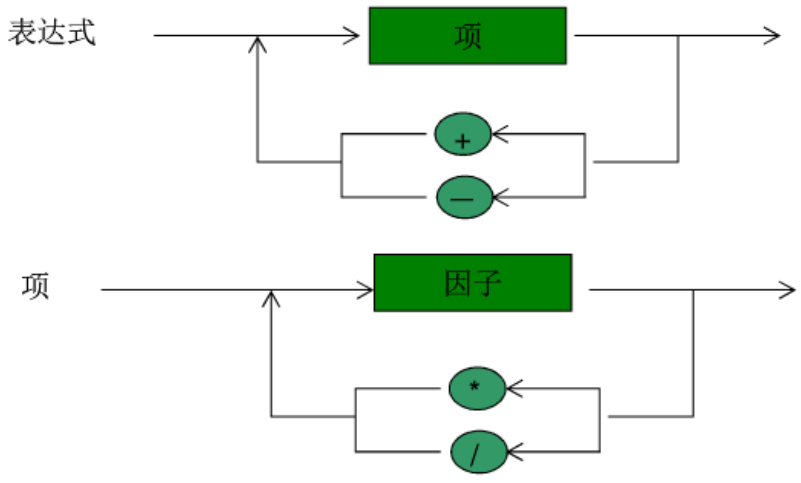
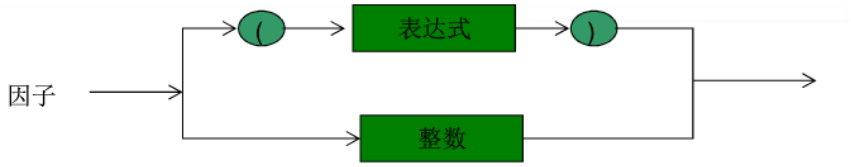
ACM基础算法

- 枚举
  - 定义
    - 枚举便是依次列举出所有可能产生的结果，根据题中的条件对所得的结果进行逐一的判断，过滤掉那些不符合要求的，保留那些符合要求的，也可以称之为暴力算法。
  - 适用条件
    - 当问题的所有解的个数不太多时，并在我们题目中可以接受的时间内得到问题的解，才可以使用枚举。
  - 代码流程
    - for、while、do-while循环即可。
- 递归
  - 定义
    - “参考递归定义”
  - 适用条件
    - 递归算法适用于每一步条件有着相同或者类似操作的算法，且下一步条件基于前一步操作所得出来的结果。
    - 递归算法的一个基本假设是之前操作与条件已经全部成立，在这个假设下进行进一步的操作。
  - 代码流程
    - 先写程序递归结束的终止条件，一般是递归到初始条件，必须跟上return XX; 作为递归结束语句。
    - 再写程序的普适性操作，对需要基于先前结果的地方进行函数的调用，注意参数的设置。
  - 特殊用法
    - 也有利用递归代替未知循环次数的循环，其时间复杂度与循环一样。
    - 此时的递归终止条件一般为设置循环此时的最后一次。
    - 例如N皇后问题
      -

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int N;
int queenPos[100];

void NQueen(int k);

int main()
{
    cin>>N;
    NQueen(N);
    return 0;
}

void NQueen(int k)
{                       //在0~k-1行皇后已经摆好的情况下，摆第k行及其后的皇后
    int i;

    if(k == N)                          //N个皇后已经摆好
    {
        for(i=0;i<N;i++)
            cout<<queenPos[i]+1<<" ";
        cout<<endl;
        return ;
    }

    for(i=0;i<N;i++)                    //逐尝试第k个皇后的位置
    {
        int j;
        for(j=0;j<k;j++)                    //和已经摆好的k个皇后的位置比较，看是否冲突
        {
            if(queenPos[j]==i||abs(queenPos[j]-i)==abs(k-j))
                break;                              //冲突，下一个位置
            if(j==k)                            //当前选的位置i不冲突
            {
                queenPos[k]=i;                                  //将第i个皇后摆放在位置i
                NQueen(k+1);
            }
        }       //for(i=0;i<N;i++)
    }
}
```

- ACM示例
  - 表达式求值
    - 题目描述
      - 输入为四则运算表达式，仅由整数、+、-、*、/、（、）组成，没有空格，要求求其值。假设运算结果符都是整数。“/”结果也是整数。
    - 样例输入
      - (2+3)*(5+7)+9/3
    - 样例输出
      - 63
    - 解题思想
      - 表达式是个递归的定义：



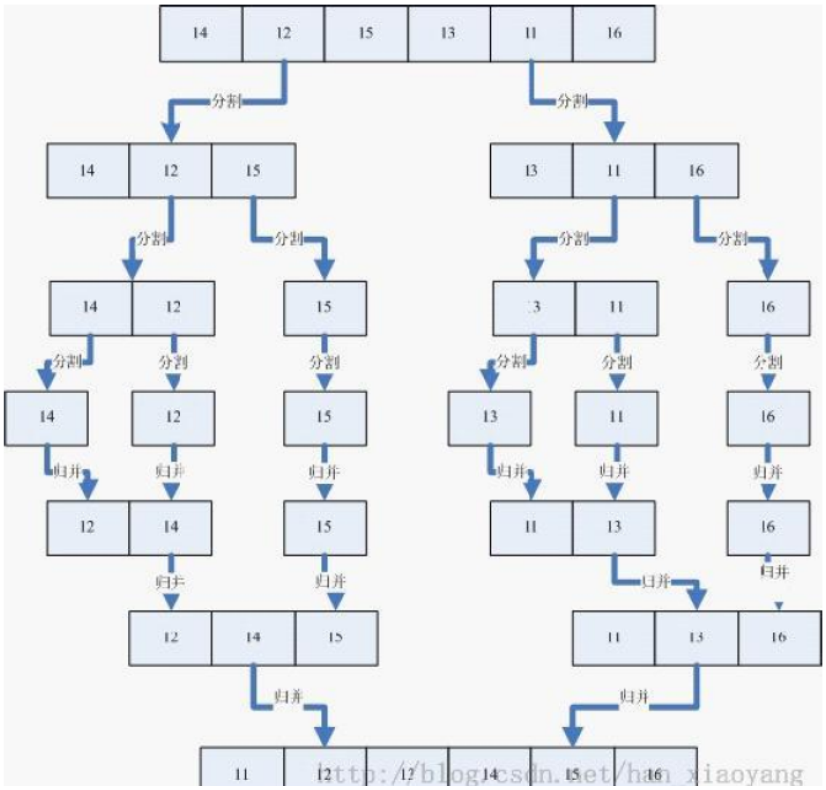


- 解题代码
  -

```cpp
#include<iostream>
#include<cstring>
#include<cstdlib>
using namespace std;
int factor_value();
int term_value();
int expression_value();

int expression_value()              //求一个表达式的值
{
    int result = term_value();              //求第一项的值
    bool more =true;
    while(more)
    {
        char op = cin.peek();           //看一个字符，不取走
        if(op == '+'|| op == '-')
        {
            cin.get();                  //从输入中取走一个字符
            int value = term_value();
            if( op == '+')
                result += value;
        }
        else more = false;
    }
    return result;
}

int factor_value()      //求一个因子的值
{
    int result = 0;
    char c = cin.peek();
    if(c == '(')
    {
        cin.get();
        result = expression_value();
        cin.get();
    }
    else
    {
        while(isdigit(c))
        {
            result = 10*result+c-'0';
            cin.get();
            c = cin.peek();
        }
    }
    return result;
}

int term_value()    //求一个项的值
```

```
{
    int result = factor_value();//求第一个因子的值
    while(true)
    {
        char op = cin.peek();
        if(op == '*'||op == '/')
        {
            cin.get();
            int value = factor_value();
            if(op == '*')
                result *= value;
            else
                result /= value;
        }
        else
            break;
    }
    return result;
}


int main()
{
    cout<<expression_value()<<endl;
    return 0;
}
```

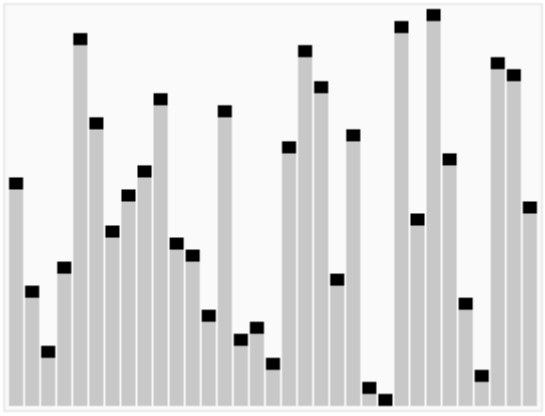
- 分治
  - 定义
    - 把一个待处理模型分解成几个与原模型形式相同的小模型，递归地解决处理完成小模型后，实现整个模型的成立（一般是合并）。
  - 适用条件
    - 该问题的规模缩小到一定的程度就可以容易地解决；
    - 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。
    - 利用该问题分解出的子问题的解可以合并为该问题的解；
    - 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。
  - 示例
    - 归并排序
      - 简介
        - 归并排序是一种稳定的算法，采用分治的思想，有序的子序列合并得到有序序列
      - 代码流程
        - 先将序列分成长度为 n/2的两部分
        - 再对于左右两部分采用分治的方法得到有序序列
        - 最后将左右两个有序序列合并得到整个有序序列



- 功能实现
  - 时间复杂度为O(n logn)

```
void arry_add(int arry[], int left, int mid, int right)     //归并
{
    if(left >= right)
        return ;
    int i = left, j = mid + 1, k = 0;
    while(i <= mid && j <= right)
    {
        if(arry[i] <= arry[j])
            tmp[k++] = arry[i++];
        else
            tmp[k++] = arry[j++];
    }
    while(i <= mid)
        tmp[k++] = arry[i++];
    while(j <= right)
        tmp[k++] = arry[j++];
    for(i = 0; i < k; i++)
        arry[i + left] = tmp[i];    //覆盖
}

void merge_sort(int arry[], int left, int right)
{
    if(left >= right)
        return ;    //分割终止条件
    int mid = (left + right)/2;
    merge_sort(arry, left, mid);    //左半分割
    merge_sort(arry, mid + 1, right);       //右半分割
    arry_add(arry, left, mid, right);       //归并
}
```

  - 详细图解：http://www.cnblogs.com/chengxiao/p/6194356.html
- 快速排序
  - 简介
    - 快速排序使用分治法策略来把一个序列分为两个子序列，是二叉查找树的一个空间最优化版本。
  - 代码流程
    - 从数列中挑出一个元素，称为"基准"。
    - 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（相同的数可以到任一边）。在这个分区结束之后，该基准就处于数列的中间位置。这个称为分区操作。
    - 递归地把小于基准值元素的子数列和大于基准值元素的子数列排序。
    -



- 功能实现
  - 一般时间复杂度O(n logn)、空间复杂度O(n)；最坏时间复杂度O(n**2)

```
void quick_sort(int arry[],int left,int right)
{
    if(left >= right)
        return ;
    int i = left, j = right;
    int tmp = arry[left];
    do
    {
        while(arry[j] > tmp && i < j)
            j--;
        if(i < j)
        {
            arry[i] = arry[j];
            i++;
        }
        while(arry[i] < tmp && i < j)
            i++;
        if(i < j)
        {
            arry[j] = arry[i];
            j--;
        }
    }while(i != j);
    arry[i] = tmp;
    quick_sort(arry, left, i-1);
    quick_sort(arry, i+1, right);
}
```

- ACM示例
  - 求排列的逆序数
    - 题目描述
    - 样例输入
    - 样例输出
    - 解题思想
    - 解题代码

幕布 - 思维概要整理工具