알고리즘

01. Divide & Conquer

2014/09/12

미디어소프트웨어학과
민경하

Contents

1. Divide & Conquer

1.0 Introduction

1.1 Multiplication

1.2 Recurrence relation

1.3 Sorting

1.4 Medians

1.5 Matrix multiplication

1.6 Finding closest pair

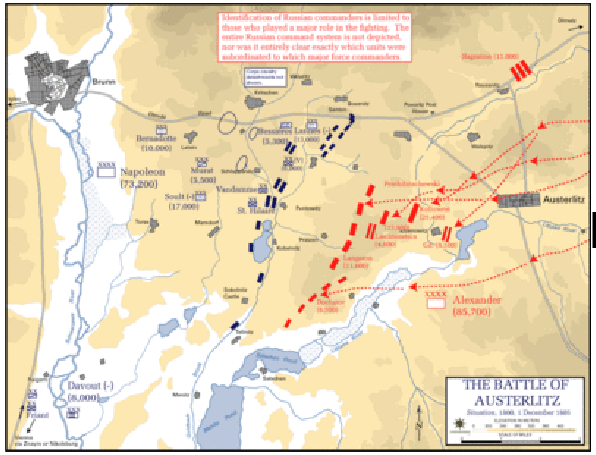
1.0 Introduction

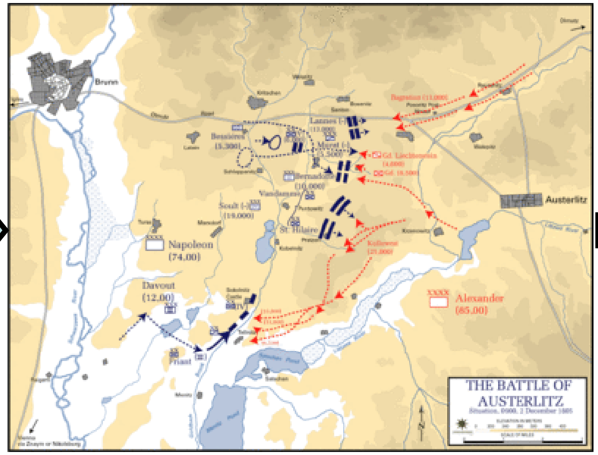
(1) Battle of Austerlitz

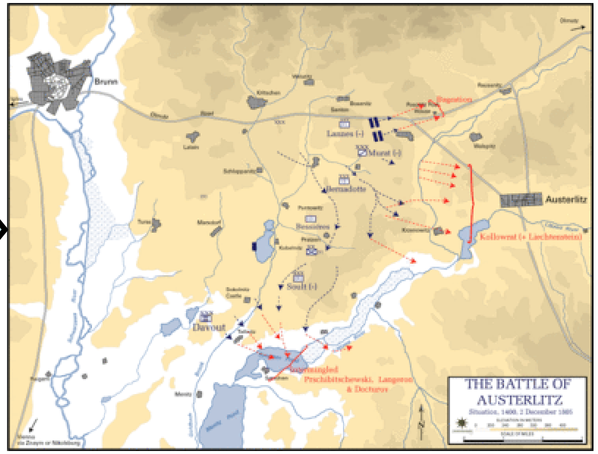
1.0 Introduction

(1) Battle of Austerlitz (1805)

- Battle of three emperors
- France (65,000) VS Austria + Russia (80,000)
- The end of the 3rd anti-France aliance







1.0 Introduction

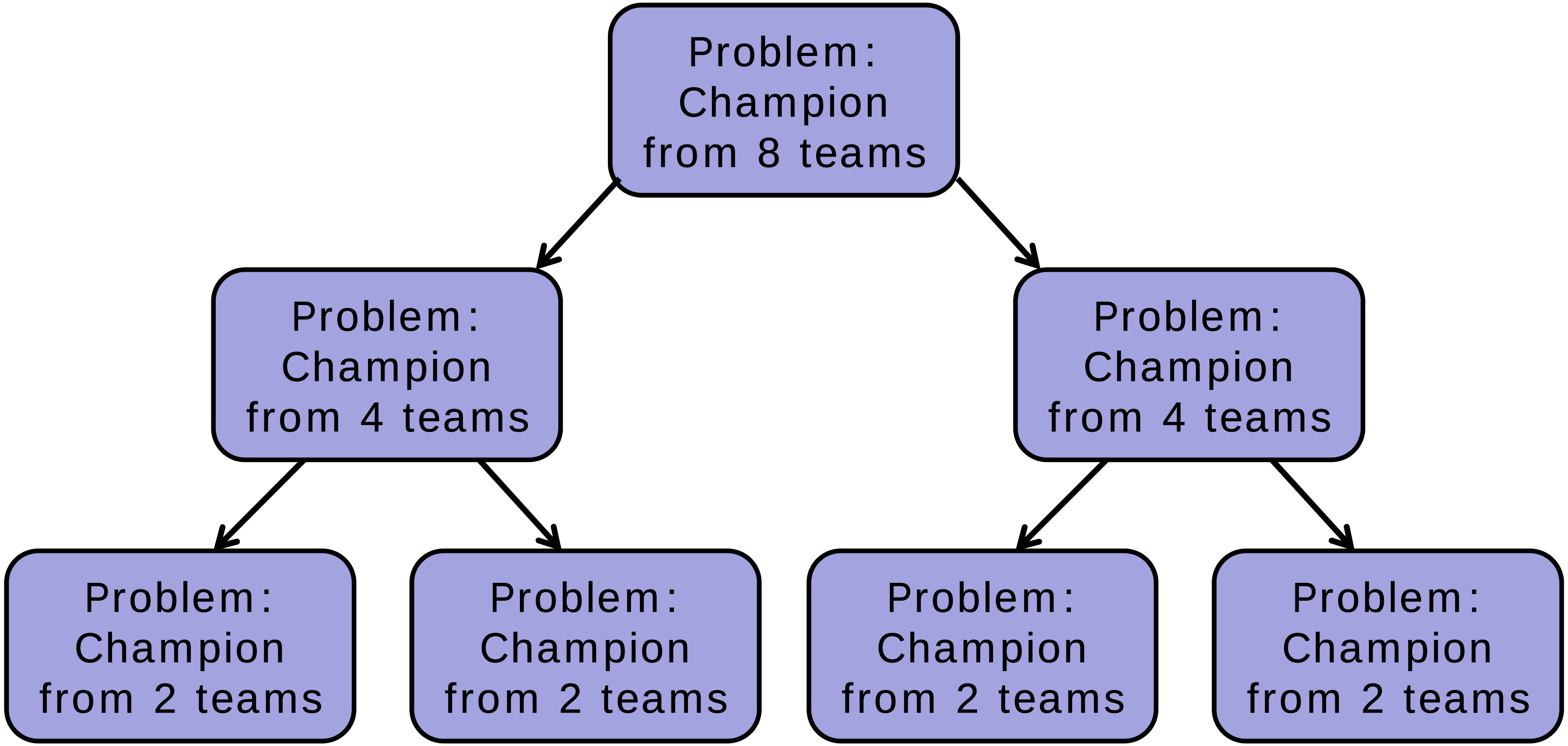
(2) League VS Tournament

- Ex) Elite-8 of Worldcup 2014
  - BRA, COL, FRA, GER, NED, CRC, ARG, BEL
- Champion by league
  - How many games do they play?
- Champion by tournament
  - How many games do they play?

1.0 Introduction

(2) League VS Tournament

– Champion by tournament



1.0 Introduction

(2) League VS Tournament

– Champion by tournament

```
int Champion8 ( {BRA, COL, FRA, GER, NED, CRC, ARG, BEL} )
{
    Lwinner = Champion4 ( {BRA, COL, FRA, GER} );
    Rwinner = Champion4 ( {NED, CRC, ARG, BEL} );

    return Winner ( Lwinner, Rwinner );
}
```

1.0 Introduction

(2) League VS Tournament

– Champion by tournament

```
int Champion4 ( {BRA, COL, FRA, GER} )
{
    Lwinner = Champion2 ( {BRA, COL} );
    Rwinner = Champion2 ( {FRA, GER} );

    return Winner ( Lwinner, Rwinner );
}
```

```
int Champion4 ( {NED, CRC, ARG, BEL} )
{
    Lwinner = Champion2 ( {NED, CRC} );
    Rwinner = Champion2 ( {ARG, BEL} );

    return Winner ( Lwinner, Rwinner );
}
```

1.0 Introduction

(2) League VS Tournament

– Champion by tournament

```
int Champion2 ( {BRA, COL} )
{
    Lwinner = Champion1 ( {BRA} );      //     Unnecessary
    Rwinner = Champion1 ( {COL} );      //     Unnecessary

    return Winner ( BRA, COL);
}
```

```
int Champion2 ( {BRA, COL} )
{
    Lwinner = Champion1 ( {BRA} );      //     Unnecessary
    Rwinner = Champion1 ( {COL} );      //     Unnecessary

    return Winner ( BRA, COL);
}
```

1.0 Introduction

(2) League VS Tournament

– Champion by tournament

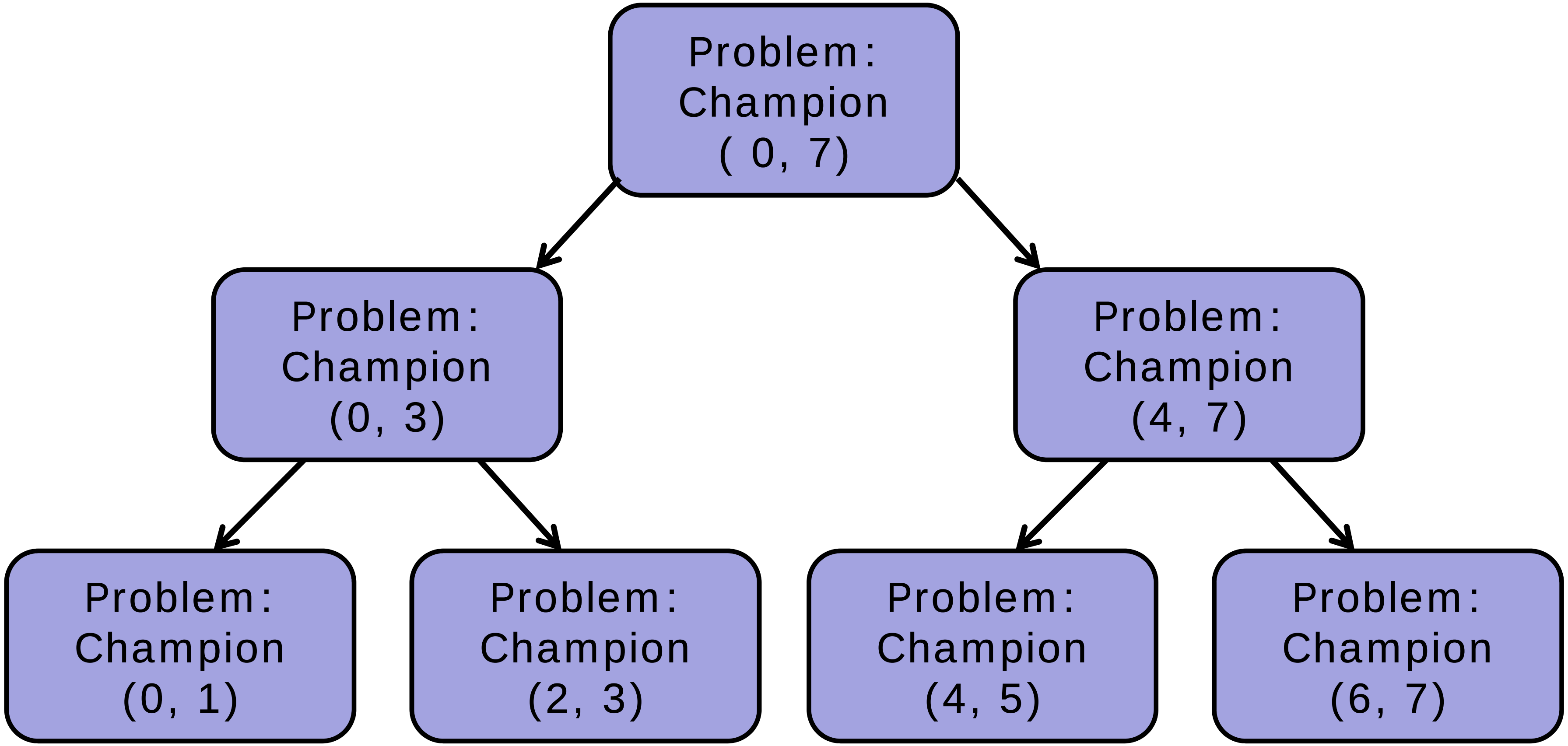
```
char sweet8[8] = {BRA, COL, FRA, GER, NED, CRC, ARG, BEL};
int Champion ( int s, int e )    //initially, s = 0, e = 7
{
    int m = (s + e)/2;
    Lwinner = Champion ( s, m );  //     s = 0, m = 3
    Rwinner = Champion ( m+1, e );//     m+1 = 4, e = 7

    return Winner ( Lwinner, Rwinner );
}
```

1.0 Introduction

(2) League VS Tournament

– Champion by tournament



1.0 Introduction

(2) League VS Tournament

– Champion by tournament

```
char sweet8[8] = {BRA, COL, FRA, GER, NED, CRC, ARG, BEL};
int Champion ( int s, int e )     //initially, s = 0, e = 7
{
    int m = (s + e)/2;
    Lwinner = Champion ( s, m ); //     s = 0, m = 3
    Rwinner = Champion ( m+1, e );//    m+1 = 4, e = 7

    return Winner ( Lwinner, Rwinner );
}
```

– Do we miss something?

1.0 Introduction

(2) League VS Tournament

– Champion by tournament

```
char sweet8[8] = {BRA, COL, FRA, GER, NED, CRC, ARG, BEL};
int Champion ( int s, int e )     //initially, s = 0, e = 7
{
    if ( s == e )
        return s;

    int m = (s + e)/2;
    Lwinner = Champion ( s, m );  //     s = 0, m = 3
    Rwinner = Champion ( m+1, e );//     m+1 = 4, e = 7

    return Winner ( Lwinner, Rwinner );
}
```

– Do we miss something? → **degenerate case**

(2) League VS Tournament

– Champion by tournament

```
char sweet8[8] = {BRA, COL, FRA, GER, NED, CRC, ARG, BEL};
int Champion ( int s, int e )    //initially, s = 0, e = 7
{
    if ( s == e )
        return s;                                         → Degenerate case

    int m = (s + e)/2;                                    → Divide

    Lwinner = Champion ( s, m );
    Rwinner = Champion ( m+1, e );                        → Conquer

    return Winner ( Lwinner, Rwinner );                   → Combine
}
```

1.0 Introduction

(3) Key idea of divide & conquer

- Solve a problem of *n* inputs by splitting the input into *k* subsets

– Three steps of divide & conquer

- Divide
  - – Breaking a problem into subproblems
- Conquer
  - – Recursively solving these subproblems
- Combine (optional)
  - – Appropriately combining their answers

1.0 Introduction

(4) Abstract algorithm for DnC (recursive)

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

– Most divide & conquer algorithms are implemented using recursive call

1.0 Introduction

(5) Three check points

- Same format
- Reduced problem size
- Degenerate case

1.0 Introduction

- Same format

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

- Reduced problem size

```
global n, A(1:n);
DnC ( int p, int q )
{
        int m;
        if ( SMALL (p, q) )
                return G (p, q);
        else
                m ← DIVIDE (p, q);
                return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

- Degenerate case

```
global n, A(1:n);
DnC ( int p, int q )
{
    int m;
    if ( SMALL (p, q) )
        return G (p, q);
    else
        m ← DIVIDE (p, q);
        return COMBINE ( DnC (p, m), DnC (m+1, q) );
}
```

1.0 Introduction

(6) Performance analysis for DnC

- T (n): time complexity of DnC ( ) for n inputs
- g (n): for small input
- f_1 (n): for DIVIDE ( )
- f_2 (n): for COMBINE ( )

$$T(n) = \begin{cases} g(n), & \text{for small } n \\ 2T(n/2) + f_1(n) + f_2(n), & \text{otherwise} \end{cases}$$

$$T(n) = \begin{cases} g(n), & \text{for small } n \\ aT(n/b) + O(n^d), & \text{otherwise} \end{cases}$$

1.0 Introduction

(7) The simplest divide & conquer algorithm

- Binary search:
  - Let $A = \{a_1, \dots a_n\}$ be a list of elements which are sorted in nondecreasing order.
  - Determine whether x is in A or not. If x is in A, then find i such that $a_i = x$.

1.0 Introduction

- Straightforward search

```
int bsearch_straightforward ( int s, int e, int A[], int x )
{
    for ( i = s; i <= e; i++ ) {
        if ( A[i] == x )
            return i;
    }

    return NONE;
}
```

  - Performance analysis?

1.0 Introduction

- Binary search (Divide & Conquer)

```
int bsearch ( int s, int e, int A[], int x )
{
    if ( s == e )
        if ( A[s] == x )
            return s;
        else
            return NONE;
    m ← (s+e)/2;
    if ( A[m] == x )
        return m;
    else if ( A[m] > x )
        return bsearch ( s, m, A, x );
    else
        return bsearch ( m + 1, e, A, x );
}
```

1.0 Introduction

- Binary search → check three key points

```
int bsearch ( int s, int e, int A[], int x )
{
    if ( s == e )
        if ( A[s] == x )
            return s;
        else
            return NONE;
    m ← (s+e)/2;
    if ( A[m] == x )
        return m;
    else if ( A[m] > x )
        return bsearch ( s, m, A, x );
    else
        return bsearch ( m + 1, e, A, x );
}
```

1.0 Introduction

- Binary search (Divide & Conquer)
  - Performance analysis

$$T(n) = \begin{cases} 1, & \text{for } n = 1 \\ T(n/2) + 1, & \text{otherwise} \end{cases}$$

$$T(n) = \log n$$