# 競技プログラミング班 活動報告書

2019年11月1日

服部　瑠斗[*1]　小村　漱一朗[*2]　中野　海人[*3]
稲垣　和真[*4]　浜田　直弥[*5]　西見　元希[*6]
石川　琉聖[*7]

[*1] 情報理工学部 情報理工学科 知能情報コース 二回生
[*2] 情報理工学部 情報理工学科 知能情報コース 三回生
[*3] 情報理工学部 情報理工学科 知能情報コース 三回生
[*4] 情報理工学部 情報理工学科 知能情報コース 三回生
[*5] 情報理工学部 情報理工学科 システムアーキテクトコース 二回生
[*6] 情報理工学部 情報理工学科 セキュリティ・ネットワークコース 二回生
[*7] 情報理工学部 情報理工学科 一回生

# 1　活動の概要

文責: 稲垣 和真

本プロジェクトは、競技プログラミングを通してプログラミングにおけるアルゴリズムの知見を深めるために発足した。競技プログラミングに用いたサイトは AtCoder である。AtCoder が開催している AtCoder Beginner Contest や班内の活動内で開催したバーチャルコンテストを通して実際に問題を解き、その解法を活動で共有し、お互いの知見を高めたり、いろいろなアルゴリズムの手法について会員が解説し、理解を深めるというものである。以上の 2 点を主として本プロジェクトは活動した。

# 2　競技プログラミングについて

文責: 稲垣　和真

競技プログラミングとは、解くべき問題が与えられ、その問題を解くプログラムを、早く正確に解き、正解数や解くまでにかかった時間などを競う競技である。具体的には以下の 5 つの要素からなる。

- **問題文**
  解くべき問題の内容が記されている。
- **制約**
  問題文で出てきた数値や文字列に、どのような制約が与えられているかが記されている。
- **入力**
  入力は、標準入力にどのような形式で文字列を与えるかが記されている。
- **出力**
  標準出力にどのような形式で与えるかの条件が記されている。
- **入出力例**
  実際にプログラムに与えられる入力と、プログラムが出力すべき文字列が記されている。

# 3　学習内容

## 3.1　アルゴリズム

文責: 服部　瑠斗

本プロジェクトでは主に以下の 2 つのアルゴリズムを学習した。

- 深さ優先探索 (dfs)
- 累積和

### 3.1.1 深さ優先探索

深さ優先探索とは、全探索を行うアルゴリズムの種類の一つである。競技プログラミングでは主に、状態の遷移が分岐するような処理の実装に用いられている。深さ優先探索の特徴として、与えられた状態の深さに注目して探索を行うという点が挙げられる。また深さ優先探索を用いるメリットとして、再帰を用いて実装した場合コードがシンプルに記述することが出来るという点が挙げられる。

Listing 1　深さ優先探索の実装例 (ABC87 C 問題)

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, n) for(int i = 0; i < (int)(n); i++)

vector<int> a1;
vector<int> a2;
int n = 0;
void chmax(int &a, int b) { if (a < b) a = b; }

int dfs(int i, bool isDowned)
{
    // 終了条件
    if(i == n-1)
    {
        if(isDowned) return a2[i];
        else return a1[i] + a2[i];
    }

    int ans = 0;

    if(!isDowned)
    {
        chmax(ans, dfs(i+1, true) + a1[i] + a2[i] );
        chmax(ans, dfs(i+1, isDowned) + a1[i] );
    }
    else chmax(ans, dfs(i+1, isDowned) + a2[i] );

    return ans;
}

int main()
{
    cin >> n;
    vector<int> tmp;
    rep(i,n*2)
    {
        int s;
```

```
38          cin >> s;
39          tmp.push_back(s);
40      }
41      rep(i,n) a1.push_back(tmp[i]);
42      rep(i,n) a2.push_back(tmp[i+n]);
43
44      cout << dfs(0, false) << endl;
45  }
```

### 3.1.2 累積和

累積和とは、計算量を圧縮するために用いられるアルゴリズムの種類の一つである。競技プログラミングでは主に、配列上の区間の総和を求める処理の実装に用いられている。累積和の特徴として、前処理を行うことによって元々の配列が保持している情報に加えて配列のある区間の総和も求めることが出来るという点が挙げられる。

Listing 2　累積和の実装例 (ABC122 C 問題)

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  #define rep(i, n) for(int i = 0; i < (int)(n); i++)
4
5  int main()
6  {
7      int N,Q;
8      string s;
9      cin >> N >> Q >> s;
10
11     vector<pair<int, int> > lr(Q);
12     rep(i,Q) cin >> lr[i].first >> lr[i].second;
13
14     // 累積和を保持する配列の生成
15     vector<int>dp;
16     dp.push_back(0);
17
18     rep(i,s.size()-1)
19     {
20         if(s[i] == 'A' && s[i+1] == 'C') dp.push_back(dp[dp.size()-1] + 1);
21         else dp.push_back(dp[dp.size()-1]);
22     }
23     rep(i,Q)
24     {
25         cout << dp[lr[i].second-1] - dp[lr[i].first-1] << endl;
26     }
27 }
```

## 3.2 競技プログラミングにおけるテクニック

文責: 西見 元希

本項では、この活動において学ぶことのできた競技プログラミングにおけるテクニックを述べる。

- **マクロの活用**

  define マクロの活用は競技プログラミングにおける提出コードの冗長性を大きく削減し、より簡潔なコードを実現する。中でも単純な繰り返しにおける rep マクロは最も頻繁に利用される。次のコードは rep マクロの定義と利用例、マクロを用いなかった場合との比較である。

Listing 3　rep マクロの例

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

#define rep(i,n) for(int (i)=0;(i)<(n);++(i))マクロを利用しない場合

//
int main(void){
  int n=10;
  vector<int> a(n);
  for(int i=0;i<n;++i)
    a[i]=i;
  for(int i=0;i<n;++i)
    cout << a[i] << endl;
}マクロを利用した場合

//
int main(void){
  int n=10;
  vector<int> a(n);
  rep(i,n) a[i]=10;
  rep(i,n) cout << a[i] << endl;
}
```

  このコードでは要素数 10 の可変長配列に要素の座標と同じ数値を格納しそれを出力しているが、rep マクロを用いることでコードが短くなり可読性が向上したのが確認できるだろう。

- **ラムダ式の活用**

  ラムダ式は関数型プログラミングにおいてよく用いられる言語機能であるが、競技プログラミングでの主流言語である C++ にもその機能があり、ソートなどの引数として極めて便利に利用することができる。次のコードはラムダ式を利用した整数列のソートの例である。

Listing 4　lambda 式の例

```
    #include<bits/stdc++.h>
    using namespace std;
```

```
3       int main(void){
4         int n=10;
5         vector<int> a(n);
6         rep(i,n) a[i]=i;降順にソート
7
8         //
9         sort(v.begin(),v.end(),
10          [](int x, int y) -> auto{return x>y;});のようにソート
11
12        //0246813579
13        sort(v.begin(),v.end(),
14          [](int x, int y) ->
15            auto {if(x%2==y%2)return x<y;
16             else if(x%2)return x<y;
17             else return x<y;});
18      }
```

このようにラムダ式を用いたソートは非常に汎用的なソートが行える。

# 4 活動で得られたもの

文責: 服部　瑠斗

本プロジェクトで得られたものは大きく 2 つに分けられる。
1つ目は、競技プログラミングの問題に対してどのアルゴリズムやテクニックを用いるかを判断する力である。具体的には、制約を見ることによって判断することが出来る。理由としては制約から入力される値の上限が分かる。上限の値が判明すればその値が入力された場合の最悪計算量 (オーダー) を大まかに求めることが出来る。最悪計算量が判明すれば、そこから全探索が出来るのか出来ないかといったアルゴリズムやテクニックの選択が可能になる。
2 つ目は、アルゴリズムやテクニックを実際のコードに落としこみ問題を解く力である。一般的にアルゴリズムやテクニックをただ学ぶだけでは実践で苦労してしまう。そこで本プロジェクトでは、問題の内容を細分化することによって実践に落としこむ方法をいくつかプロジェクトメンバー間で共有することによって問題を解く力を習得した。

# 5 問題点

文責: 西見 元希

このプロジェクトにおける問題点として、まず集まりの悪さが挙げられる。そもそも活動時間を班員全員が問題なく出席できる時間に設定することができなかったために班員が十分に出席できず、結果として活動は数回のバーチャルコンテストと解説にとどまることになってしまった。これは今後のプロジェクト活動において改善すべき問題点であると考えられる。具体的な解決案としては現在いくつかのプロジェクトで見られる週数回の活動日を設定し、同じ内容を学習するというものが挙げられる。

# 6　展望

文責: 服部　瑠斗

今回の活動では、基礎的なアルゴリズムやテクニックの学習やバーチャルコンテストを用いた実践の体験を行った。今後の展望としては、活動で扱ったアルゴリズムなどを組み合わせた新しいアルゴリズムを用いて問題を解く力を養う為の活動を行いたいと考えている。

# 参考文献

技プログラミングを知ろう！ [第 1 回] : https://book.mynavi.jp/manatee/detail/id=56242