ABC 088 の解説を書いてみる

AtCoder

初めて全完できたので

Tasks - AtCoder Beginner Contest 088

A問題

500円玉を無限に使えて、1円玉の数Aが与えられるとき、金額Nを払えるかどうかを出力。

解法

500円玉はいくらでも使えることから、500円玉で払えるだけ払った後の金額は、500で割った余りになる。あとはその余りと1円玉の枚数Aの大小を比較すれば良い。

```
int main() {
    int n,a;
    cin>>n>>a;
    n%=500;
    if(n<=a)cout<<"Yes"<<endl;
    else cout<<"No"<<endl;

    return 0;
}
```

B問題

カードN枚の情報が与えられる。AliceとBobは交互に好きなカードを一枚ずつ取ることができる。二人共が自分のスコアを最大化するように行動したとき、AliceはBobより何点多く取ったか求めよ。

解法

N枚の中から好きなカードを取ることができ、スコアを最大化したいので、場にあるカードの中から一番値が大きいものを取っていけば良いことになる。つまり、値をソートしてから、偶数番目の和と奇数番目の和を比較することになる。

下のコードでは、値を昇順にソートしてから配列自体を反転しているので、実質降順ソートをしている。その後ループで素直に前から配列を見ている。交互にカードを取っていくので、ループ変数iの値が偶数か奇数かによって、AliceとBobの点数を保持する変数a,bのどちらに足していくかを決める。

先手がAliceなので、点数は必ずAliceの方が大きくなる。よって求める答えは必ず正になることも確認しておこう。

※配列の要素の反転には**reverse(反転したい範囲のイテレータ)**を利用している。今回は全ての要素を反転したいので、reverse(v.begin(),v.end())とする。

```
int main() {
    int n;cin>>n;
    vector<int> v(n);
    rep(i,0,n)cin>>v[i];
    //ソートする
    Sort(v);
    //反転する
    reverse(all(v));
    int a=0,b=0;
    //前から配列を見て、i の偶奇によってどちらに足すかを決める。
    rep(i,0,n){
        if(i%2==0)a+=v[i];
        else b+=v[i];
    }
    //差を出力
    cout<<a-b<<endl;

    return 0;
}
```

C問題

要素a1,a2,a3,b1,b2,b3と３＊３のグリッドがあり、その要素ci_jはai+bjになっているという。このことが正しいかどうか検証しよう。

解法

この問題は他の人の方がよっぽどうまくやっていると思うので、あくまで僕の当時の解法を書きます。
aiやbjの値は100までなので、aiかbjのどちらかをループで全探索することを考えれば、10^6で間に合う。
ここで、

```
c1_1=a1+b1
c2_1=a2+b1
.........
c3_3=a3+b3
```

より、

```
b1=c1_1-a1
b1=c2_1-a2
b1=c3_1-a3
```

のように、b1,b2,b3それぞれについて３つの式を立てることができる。
これら3つの式が等しければ、情報が正しいことになる。ここで、a1,a2,a3を３重ループで回して全探

索して、その中でどれか一つでもb1,b2,b3のそれぞれの３つの式が一致すれば、YESと出力、一致しなければNOになる。

いや、肝心のコードはね、まあね

```
int main() {
    int v[3][3];
    rep(i,0,3)rep(j,0,3){
        cin>>v[i][j];
    }
    bool f=true;
    rep(i,0,101){ // i がa1の値
        rep(j,0,101){  // j がa2の値
            rep(k,0,101){  // k がa3の値
        //b1,b2,b3について調べる
                if((v[0][0]-i==v[0][1]-j&&v[0][1]-j==v[0][2]-k)&&(v[1][0]-i==v[1][1]-j&&v
                    cout<<"Yes"<<endl;
                    return 0;
                }
            }
        }
    }
    cout<<"No"<<endl;
    return 0;
}
```

D問題

H*Wの迷路があり、「通れるマス」と「壁のマス」がある。すぬけ君は、迷路を攻略する前に任意の「通れるマス」を「壁のマス」に変えることができ、このとき変えたマスの数がスコアになる。
スコアの最大値を求めよ。そもそもゴールできないときは、－１を出力しよう。

解法

問題の設定は、事前にマスを変化させてから迷路を攻略するというものだが、この順は逆でも良い。つまり、できるだけ通るマスが少なくなるように移動し、ゴールした後で、自分が通らなかったマスを全て壁に変えるということだ。「できるだけ通りマスを少なくする」ということはゴールまでの最短距離の問題に帰着できるので、幅優先探索（BFS）を作って

今回の幅優先探索は、ある座標の上下左右の4近傍を、スタートマスからの最短距離を更新しながら順番に見ていくものである。スタートの座標を初期位置とし、距離を０とする。次にその上下左右のマスを見て、存在する座標か、壁マスかどうか、まだ訪れていないかの3つの要素を確認しながら調べて行く。これらの条件を全て満たしていれば、そのマスの最短距離を保存し、またその上下左右のマスを調べる・・ということを右下のマスに到達するまでやる。

このようにしてゴールまでの最短距離を求めることができる。この距離はすぬけ君が通るマスの数と一致しており、それ以外の'.'のマスを壁'#'に変えることでスコアは最大になる。
事前に配列に格納する時点で、'.'の数を数えておこう。

```
int main() {
    int h,w;
    cin>>h>>w;
    vector<vector<char>> v(h,vector<char>(w));
    int a=0,b=0;
    rep(i,0,h){
        rep(j,0,w){
            cin>>v[i][j];
            if(v[i][j]=='#')a++;
            else b++;
        }
    }
    //最短距離保存用の配列（非常に大きい値で初期化）
    vector<vector<int>> d(h,vector<int>(w,INF));
    //どの座標を確認したら良いかを保存するキュー
    queue<P> que;
   //とりあえず左上の座標を見るのでpush
    que.push(P(0,0));
    //左上の最短距離はもちろん０
    d[0][0]=0;

  //キューの要素がある間（見る座標の候補がある間）
    while(que.size()){
        P p=que.front(); que.pop();
   //右下のマスにたどり着けたら終わる
        if(p.first==h-1&&p.second==w-1)break;

        rep(i,0,4){
            int nx=p.second+dx[i],ny=p.first+dy[i];
            //存在する座標か、壁マスかどうか、まだ訪れていないかの3つの要素を確認
            if(0<=nx && 0<=ny && nx<w && ny<h && v[ny][nx]!='#' && d[ny][nx]==INF){
                que.push(P(ny,nx));
                d[ny][nx]=d[p.first][p.second]+1;
            }
        }
    }
  //値が非常に大きいままならそもそも右下に到達できてないので、-1
    if(d[h-1][w-1]==INF)cout<<-1<<endl;
    else cout<<(b-d[h-1][w-1]-1)<<endl;
```

```
    return 0;
}
```

ごつちやん (id:gotutiyan) 2年前