ACM ICPC2016 国内予選　問題 D　ダルマ落とし

http://icpc.iisf.or.jp/past-icpc/domestic2016/problems/all_ja.html#section_D

- 最も多くのブロックを取り除いた状況を考えると、取り除かれたブロックは、いくつかの連続ブロック群から成る。したがって、取り除くことができる連続ブロック群を全て求め、それらの重複しない組み合わせで最も多くのブロックを取り除けるケースを見つければ良い。
- ブロック数 n に対して取り除き可能な連続ブロック群は ${}_nC_2$ 通りの可能性があるが、n は 300 以下なので ${}_nC_2 \leqq 44{,}850$ である。
- 取り除き可能な連続ブロック群 $G=B_iB_{i+1}\cdots B_{j-1}B_j$ があるとする。このとき
  - G の両端に隣接するブロック B_{i-1} と B_{j+1} の重みの差が 1 以下ならこれらのブロックも取り除けるので、$G' = B_{i-1}\,G\,B_{j+1}$ も取り除き可能な連続ブロック群となる。
  - G に隣接する取り除き可能な連続ブロック $G' = B_{i'}\cdots B_{i-1}$ や $G''=B_{j+1}\cdots B_{j'}$ が存在するならば、$G'\,G$ や $G\,G''$ も取り除き可能な連続ブロックとなる。
- まず全ての隣接ブロック B_iB_{i+1} に対し取り除き可能なもの（重みの差が 1 以下）をキューに入れる。キューから取り出した連続ブロック群に対し、上記の処置で新たな連続ブロックが見つかればそれをキューに入れる。これをキューが空になるまで繰り返すことにより、全ての取り除き可能な連続ブロック群を求めることが出来る。
- ある区間[i,j]$(B_i\cdots B_j)$で取り除くことが出来るブロック数の最大値を max(i,j)とする。
  - [i,j]が取り除き可能な連続ブロック群なら $\max(i,j) = (j - i + 1)$ である。
  - そうでない場合、[i,j]に真に含まれる取り除き可能な連続ブロック群を[k,m]とすると, $\max(i,j) =$ 全ての[k,m]に対する $\max(i,k-1) + (m - k + 1) + \max(m+1,j)$ の最大値　となる。
- 上記 max(i,j)はメモ化再帰で求めることが出来、max(0, n-1)が求める答えとなる。

【pd1.c】

```
/*************************************************************
 * ACM ICPC2016 国内予選 問題 D ダルマ落とし
 * Filename:      pd1.c
 * Compile:       cc -Wall -Ofast pd1.c
 * Check:         ./a.out < D0 > D0.result; diff D0.ans D0.result
 * Algorithm:
 *   ・落とせるブロック数が最大になる解が求まったとすると
 *     落とせるブロックは、複数の重複しない連続ブロックの和集合とみなせるので
 *     a) まず落とすことが可能な全ての連続ブロックを求め
 *     b) 次にそれらの重複しない組み合わせでブロック数が最大となるものを探索する
 *   [a)の処理]
 *       隣接ブロックで同時に落とせるものを探しキューに入れる
 *       キューから連続ブロックを１個取り出し
 *       その両サイドのブロックが同時に落とせるなら
 *       両サイドに広げた連続ブロックをキューに入れ、
 *       また、隣接する連続ブロックがあればそれを連接してキューに入れる
 *       この処理をキューが空になるまで繰り返す
 *   [b)の処理]
 *       区間[i,j]の範囲で落とせる最大のブロック数は
 *          ・[i,j]が a)で求まる連続ブロックなら、これが最大
 *          ・そうで無い場合、[i,j]に含まれる各連続ブロック[k,m]に対し
 *            [i,k-1]の最大ブロック数+(m-k+1)+[m+1,j]の最大ブロック数を計算し
 *            その最大値を[i,j]の最大ブロック数とすれば良い
 *   ※ 少し遅いので最適化コンパイルする
 *************************************************************/

#include <stdio.h>
#define MAXN 300            // ブロックの最大個数
// ブロック x と y の重みの差が１以下(同時に落とせる)かどうかを判定するマクロ
#define CONNECTED(x,y) (((w[x]>w[y])?w[x]-w[y]:w[y]-w[x]) <= 1)
typedef struct {
      int from;
      int to;
} area;                         // 連続ブロック領域を表す構造体
int w[MAXN];                    // 各ブロックの重み
area queue[MAXN*MAXN/2];   // 連続ブロック領域のキュー
int qtop, qlast, qcount;
int n;                          // ブロックの個数
char ca[MAXN][MAXN];       // 領域[i,j]のブロックを全て落とせる時 ca[i][j]=1
int max[MAXN][MAXN];       // 領域[i,j]で落とすことができるブロック数の最大値

// 領域[from,to]内で落とすことができるブロック数の最大値を求める
int findmax(int from, int to)
{
      int i, j;
      int ret = 0;
      int val;
      if(to <= from) return 0;      // 領域内にブロックが無いので 0 を返す
      if(max[from][to]>=0)            // すでに最大値が求まっているので
            return max[from][to];     // その値を返す（メモ化再帰）
//      printf("%d:%d-%d;¥n", level, from, to);

      for(i=from; i<to; i++){
```

```
            for(j=to; j>i; j--){
                if(ca[i][j]){      // [i,j]は[from,to]内のブロックを落とせる連続領域
                    if(i==from && j==to){     // [i,j]=[from,to]なら最大値なので
                        max[from][to] = to - from + 1;          // 最大値を記憶して
                        return max[from][to];           // その値を返す
                    }
                    // 左右の領域の最大値の和と[i,j]のブロック数の和を求める
                    val = findmax(from, i-1) + (j - i + 1) + findmax(j+1, to);
                    // それが現時点での最大値 ret を超えるなら ret を更新
                    if(val > ret) ret = val;
                }
            }
        }
        max[from][to] = ret;            // メモ化再帰のために求まった結果を保存
        return ret;
}

// 初めての領域[from,to]をキューに入れる
int enqueue(int from, int to)
{
        area new;
        if(ca[from][to]==0){            // 過去にキューに入れたことが無いなら
            ca[from][to] = 1;           // 連続領域として登録
            max[from][to] = to - from + 1;   // [from,to]のブロックは全て落とせる
            new.from = from;             // キューへの登録データを作成し
            new.to = to;
            queue[qlast++] = new;     // キューに入れる
            qcount++;
//            printf("%d-%d;¥n", from, to);
            return 1;
        }
        return 0;
}

int main(void)
{
        int i, j;
        area old;
        while(1){
            scanf("%d", &n);            // n の入力
            if(n==0) break;             // n が 0 なら終了
            for(i=0; i<n; i++)
                scanf("%d", &w[i]); // 各ブロックの重みの入力
            for(i=0;i<n; i++)
                for(j=0; j<n; j++){
                    ca[i][j] = 0;     // 落とせる連続領域なしと初期化
                    max[i][j] = -1; // 領域[i,j]の落とせるブロック数未定と初期化
                }
            qtop = qlast = qcount = 0;          // キューの初期化
            // 隣接するブロックが同時に落とせるならキューに入れる
            for(i=0; i<n-1;i++){
                if(CONNECTED(i, i+1)){          // ブロック i と i+1 は同時に落とせる
                    enqueue(i, i+1);            // 領域[i, i+1]をキューに入れる
                }
```

```
            }
            while(qcount){                          // キューが空でない間繰り返す
                old = queue[qtop++];                // キューから一つ領域を取り出し old とする
                qcount--;
                if(old.from > 0 && old.to < n-1){    // old の前後にブロックがあり
                    if(CONNECTED(old.from - 1, old.to + 1)){ // それを同時に落とせる
                        enqueue(old.from-1, old.to+1); // 前後に広げた領域をキューに
                    }
                }
                if(old.from >= 2){                  // 前(左)に 2 個以上のブロックがあるなら
                    for(i=0; i<old.from-1; i++){    // 全ての
                        if(ca[i][old.from-1]){      // 左隣接連続領域を連接して
                            enqueue(i, old.to);     // キューに入れる
                        }
                    }
                }
                if(old.to < n-2){                   // 後(右)に 2 個以上のブロックがあるなら
                    for(i=n-1; i>old.to+1; i--){    // 全ての
                        if(ca[old.to+1][i]){        // 右隣接連続領域を連接して
                            enqueue(old.from, i);   // キューに入れる
                        }
                    }
                }
            }
            // 求まった連続領域を用いて全体で落とせるブロック数の最大値を求める
            printf("%d¥n", findmax(0, n-1));
        }
        return 0;
}
```