平成26年度 茨城大学大学院 理工学研究科
博士前期課程 入学試験問題

情報工学専攻

アルゴリズムとプログラミング

注意事項

1. 試験開始の合図があるまで，この問題冊子の中を見てはいけません．
2. この試験に使用する解答用紙は ５枚です．枚数を確認して，解答用紙の表題の右側にある（ ）内に，(その１）から（その５）までを記入してください．
   問題１の問１，問２に対し（その１）を，
   問題１の問３，問４，問５に対し（その２）を，
   問題２の問１に対し（その３）を，
   問題２の問２に対し（その４）を，
   問題２の問３，問４，問５に対し（その５）を使用してください．
3. 解答用紙の全てのページに，解答科目名と受験番号を記入してください．
   氏名欄は記入しないでください．受験番号に記入漏れや記入間違いがあると，試験は無効になりますので，手元の受験票で確認のうえ注意深く記入してください．
   【上記の２と３に記載の事項は，試験開始前に必ず記入してください．】
4. 試験中に問題冊子の印刷不鮮明，ページの落丁・乱丁等に気が付いた場合は，手を上げて試験監督に知らせてください．
5. 携帯電話の電源は必ず切ってカバン等に入れ，身に着けないでください．また，携帯電話を時計として使用することはできません．
6. 試験中に辞書類・電卓を使用することはできません．

英語対応表

ヒープソート	heap sort
ヒープ	heap
整列	sort
親ノード	parent node
子ノード	child node
木	tree
配列	array
要素	element
根	root
多角形	polygon
平面	plain
相異なる	distinct
折れ線	chain of straight line segments
囲まれた	enclosed
図形	figure
交差	intersect
接触	touch
内角	interior angle
未満	less than
凸	convex
凹	concave
整数値	integer value
角度	angle
度数	degree

アルゴリズムとプログラミング

問題 1　ヒープソートとはヒープを用いた整列法である．ヒープとは親ノードの値が必ず子ノードの値以上となるが，同じ親ノードをもつ子ノードの間では値の大小関係が決められていない木で，ここでは配列を用いて実装するものとする．ヒープの例を図 1 に，図 1 のヒープを配列 d を用いて実装した例を図 2 に示す．

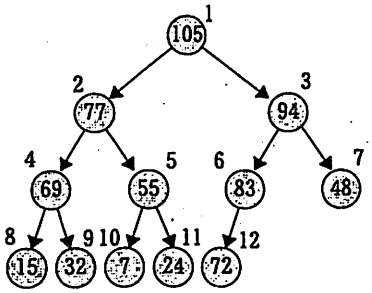


図 1 ：ヒープの例

d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]	d[8]	d[9]	d[10]	d[11]	d[12]
105	77	94	69	55	83	48	15	32	7	24	72

図 2 ：図 1 のヒープを配列で実装した例

ヒープを実装した配列の最初の要素はヒープの根に対応し，その値は配列全体の最大値である．最初の要素は最大 2 つの子を持ち，それは 2 番目の要素と 3 番目の要素である．それらの値は最初の要素の値以下となる．

2 番目の要素も最大 2 つの子を持ち，それらは 4 番目の要素と 5 番目の要素である．それらの値は 2 番目の要素の値以下となる．

3 番目の要素も最大 2 つの子を持ち，それらは 6 番目の要素と 7 番目の要素である．それらの値は 3 番目の要素の値以下となる．

以下同様に，i 番目の要素は最大 2 つの子を持ち，それらは $2i$ 番目の要素と $2i+1$ 番目の要素である．それらの値は i 番目の要素の値以下となる．

ヒープソートはこのような性質を持つヒープを用いて以下の 2 つの手順で整列を行う．

手順 1 整列対象の配列をヒープの条件を満たす並びへ変換する．

手順 2 配列の最初の要素と最後の要素の値を入れ替え，最後の要素を除いた部分配列をヒープの条件を満たす並びへ再変換する．これを部分配列の要素数が 1 になるまで繰り返す．

この手順 1, 手順 2 は，後述の手順 A を用いると以下の手順 1', 手順 2' によって実現できる．

手順 1'

1-1 配列の要素数 n に対し，$i = n/2$(小数点以下は切り捨て) を求める．

1-2 i が 1 以上であれば，1-3 から 1-4 を実行する．

1-3 配列の i 番目の要素から n 番目の要素を手順 A で並べ替える.

1-4 i を 1 減じ, 1-2 に戻る.

手順 2'

2-1 配列の要素数 n に対し, $i = n$ とする.

2-2 i が 2 以上であれば, 2-3 から 2-5 を実行する.

2-3 配列の最初の要素と i 番目の要素を交換する.

2-4 配列の最初の要素から $(i-1)$ 番目の要素を手順 A で並べ替える.

2-5 i を 1 減じ, 2-2 に戻る.

手順 A は *start* と *end* の 2 つの値が与えられたときに, *start* 番目の要素から *end* 番目の要素からなる部分配列をヒープの条件を満たす並びへ変換する手続きである. ただし, この部分配列は *start* 番目の要素のみが子要素のどちらかあるいは両方より値が小さいことがあるが, 他の親子間では必ず親要素の値が子要素の値以上となっている, すなわち *start* 番目の要素以外はヒープの条件を満たしているものとする.

手順 A

a-1 $i = start$ とする.

a-2 親要素 (i 番目の要素) に子要素 ($2i$ 番目の要素) が存在するならば (すなわち $2i \leq end$ ならば), a-3 から a-5 を実行する.

a-3 親要素と子要素の値を比較する. 子要素が 1 つしかない場合 ($2i + 1 > end$ のとき) にはその子要素 ($2i$ 番目の要素) と比較を行うが, 子要素が 2 つの場合 ($2i + 1 \leq end$ のとき) には, 子要素同士の値を比較し, 値の大きい方と比較を行う.

a-4 親要素の値が子要素の値以上であれば手順 A を終了する. 子要素の方が値が大きければ, 親要素の値と子要素の値を交換して a-5 へ進む.

a-5 値を交換した方の子要素を親要素とし (すなわち $i = 2i$ または $i = 2i + 1$ とし), a-2 に戻る.

リスト 1 は整数配列に対するヒープソートを実現する C 言語のプログラムの一部である. 関数 downheap は手順 A を実現する関数であり, 引数 d は整列対象となる配列, start と end は並べ替えの範囲を指定する要素番号 (ただし start ≤ end) とする. 関数 heapsort はヒープソートを実現する関数であり, 引数 d は整列対象の配列, n は配列の要素数とする.

リスト 1 の空欄部分にはそれぞれ手順 A, 手順 1', 手順 2' を実装するプログラムが記述されるものとする.

なお, 本問題の説明では, 配列の要素番号を 1 から数えているが, C 言語では配列の要素番号は 0 から始まる. 配列の 0 番目の要素を使用すると, i 番目の要素の子が $2i + 1$ 番目と $2i + 2$ 番目となり, 直感的に理解しにくくなるため, 本問題では配列の 0 番目の要素を未使用とし, 1 番目の要素から使用するものとする. したがって, 本問題で {1, 2, 3, 4} のように表記している配列は, 実際には {0, 1, 2, 3, 4} のようになり, 0 番目に未使用の要素が存在することになる.

以下の問 1〜問 5 に答えよ.

問 1 配列 { 6, 99, 73, 55, 36, 60, 16 } の 1 番目の要素から 7 番目の要素を手順 A で並べ替えたときの値の並びを示せ.

問 2 手順 A に従い，リスト 1 の関数 downheap のプログラムを完成させよ．必要な変数は自由に宣言して使用してよい．

問 3 配列 {25, 3, 83, 53, 13, 23, 95, 7, 31, 43, 67, 49, 61, 1, 91} を手順 1' で並べ替えたときの値の並びを示せ．

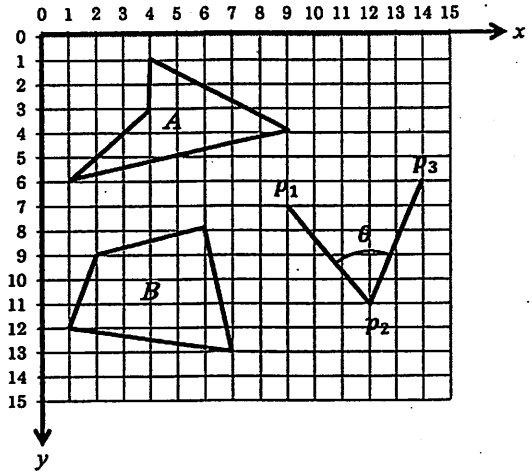
問 4 リスト 1 の該当箇所に手順 1' に相当する C 言語のプログラムを記述せよ．変数 i，tmp は必要に応じて自由に使用してもよい．

問 5 リスト 1 の該当箇所に手順 2' に相当する C 言語のプログラムを記述せよ．変数 i，tmp は必要に応じて自由に使用してもよい．

リスト 1 ：ヒープソートを実現するプログラム

```
void downheap(int d[], int start, int end) {
    手順A（問2）
}

void heapsort(int d[], int n) {
    int i, tmp;

    手順1'（問4）

    手順2'（問5）

    return;

}
```

問題２　多角形とは，平面上に与えられた相異なる点の列を折れ線で順に結んで行き，最後に，最後の点と最初の点と結んで囲まれる図形のことである．ただし，折れ線の一部が折れ線自身の別の場所と交差するか接触する場合，その図形は多角形ではないものとする．また，すべての内角の大きさが 180 度未満であるような多角形を凸多角形，そうでない多角形を凹多角形という．下図の図形 A は，点の座標の列 $(4,1),(4,3),(1,6),(9,4)$ によって定まる凹多角形であり，図形 B は，点の座標の列 $(2,9),(1,12),(7,13),(6,8)$ によって定まる凸多角形である．点の座標の列 $(2,9),(1,12),(6,8),(7,13)$ によって定まる図形は，折れ線が交差するので多角形ではない．



リスト２は，入力された３つ以上の点の座標の列に対し，それによって囲まれる図形が多角形であるかどうか，更に多角形であるならば凸多角形であるかどうかを判定するプログラムの一部である．実行例に示すように，最初に点の数が整数値で入力され，それぞれの点が x 座標と y 座標の２つの整数値で入力されたとき，判定結果を出力する．関数 isCrossing は，p1 と p2 を端点とする線分と p3 と p4 を端点とする線分が交差か接触しているかどうかを判定する関数である．線分が交差か接触している場合には１を返し，そうでない場合は０を返す．関数 getAngle は，p1 と p2 を端点とする線分と p2 と p3 を端点とする線分が作る角度を返す関数である．上図に示すように，p1→p2→p3 の順にたどって左側にできる角度 θ を０以上 360 未満の度数で返す．

リスト２のプログラムについて，以下の問１～問５に答えよ．

問１　入力された点の座標の列によって囲まれる図形が多角形であるかどうか（点の座標がすべて相異なり，折れ線の一部が折れ線自身の別の場所と交差や接触していないかどうか）を判定する関数 isPolygon の中身を記述せよ．
(ヒント) 端点を共有する線分同士は接触している．折れ線中の端点を共有しないすべての線分の組み合わせについて，交差や接触していないかどうかを確認することになる．

問２　入力された点の座標の列によって囲まれた図形が多角形であるとき，多角形が凸多角形であるかどうかを判定する関数 isConvex の中身を記述せよ．
(ヒント) すべての内角の大きさが 180 度未満であることを確認すればよいが，点の列が時計回りの場合と反時計回りの場合を考慮する必要がある．

問３　“3 0 0 10 0 0 10” が入力されたときのプログラムの出力を書け．

問４　“4 0 10 30 10 20 0 50 20” が入力されたときのプログラムの出力を書け．

問５　“5 7 12 2 11 8 7 0 3 9 0” が入力されたときのプログラムの出力を書け．

リスト２：凸多角形かどうか判定するプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct point {
    int x;    /* x 座標の値 */
    int y;    /* y 座標の値 */
};

struct polygon {
    int n;    /* 点の個数 */
    struct point *points; /* 点の座標の列へのポインタ */
};

int isCrossing(struct point p1, struct point p2, struct point p3, struct point p4);
double getAngle(struct point p1, struct point p2, struct point p3);
int isPolygon(struct polygon p);
int isConvex(struct polygon p);

int main() {
    int i;
    int n;
    struct polygon poly;

    scanf("%d", &(poly.n));

    if (poly.n < 3) {
        printf("頂点数が足りません. \n");
        return 1;
    }

    poly.points = (struct point *) malloc(sizeof(struct point) * poly.n);

    for (i = 0; i < poly.n; i++) {
        scanf("%d %d", &(poly.points[i].x), &(poly.points[i].y));
    }

    if (!isPolygon(poly)) {
        printf("多角形ではありません. \n");
    } else if (!isConvex(poly)) {
        printf("凸多角形ではありません. \n");
    } else {
        printf("凸多角形です. \n");
    }

    free(poly.points);

    return 0;
}
```

(次のページに続く)

```
int isCrossing(struct point p1, struct point p2, struct point p3, struct point p4) {
    if (p1.x >= p2.x) {
        if ((p1.x < p3.x && p1.x < p4.x) || (p2.x > p3.x && p2.x > p4.x)) {
            return 0;
        }
    } else {
        if ((p2.x < p3.x && p2.x < p4.x) || (p1.x > p3.x && p1.x > p4.x)) {
            return 0;
        }
    }
    if (p1.y >= p2.y) {
        if ((p1.y < p3.y && p1.y < p4.y) || (p2.y > p3.y && p2.y > p4.y)) {
            return 0;
        }
    } else {
        if ((p2.y < p3.y && p2.y < p4.y) || (p1.y > p3.y && p1.y > p4.y)) {
            return 0;
        }
    }
    if (((p1.x - p2.x) * (p3.y - p1.y) + (p1.y - p2.y) * (p1.x - p3.x)) *
        ((p1.x - p2.x) * (p4.y - p1.y) + (p1.y - p2.y) * (p1.x - p4.x)) > 0) {
        return 0;
    }
    if (((p3.x - p4.x) * (p1.y - p3.y) + (p3.y - p4.y) * (p3.x - p1.x)) *
        ((p3.x - p4.x) * (p2.y - p3.y) + (p3.y - p4.y) * (p3.x - p2.x)) > 0) {
        return 0;
    }
    return 1;
}

double getAngle(struct point p1, struct point p2, struct point p3) {
    double r;
    r = (atan2(p3.y - p2.y, p3.x - p2.x) - atan2(p1.y - p2.y, p1.x - p2.x));
    r = r / M_PI * 180;  /* 弧度法を度数法に変換 */
    if (r < 0) r += 360; /* 0から360の範囲になるように調整 */
    if (r == 360) r = 0; /* ちょうど360になったときは0にする */
    return r;
}

int isPolygon(struct polygon p) { [ 問1 ] }

int isConvex(struct polygon p) { [ 問2 ] }
```