KATHOLIEKE UNIVERSITEIT LEUVEN

# Combinatorial Optimization and Local Search Techniques

Erik Demeulemeester

# Solution to HW3

```
#pragma once

class ListItem {
public:
		ListItem* getNext();
		double distance(ListItem *other);
		void setNext(ListItem *next);
		ListItem(double x, double y);
		~ListItem();

protected:
		ListItem *next;
		double x, y;



public:
		double getX(void);
		double getY(void);
};
```

# Solution to HW3

```
ListItem::ListItem(double x, double y) {
        this->x = x;
        this->y = y;
        this->next = 0;
}

ListItem::~ListItem() {
        if (next != 0) delete next;
}

void ListItem::setNext(ListItem *next) {
        this->next = next;
}
```

# Solution to HW3

```
double ListItem::distance(ListItem *other) {
        return (double)sqrt (pow(x - other->x, 2)
                + pow(y - other->y, 2));
}

ListItem* ListItem::getNext() {
        return this->next;
}
```

# Solution to HW3

```
int checkFrom(ListItem *item, ListItem *other) {
        int count = 0;
        while (other != 0) {
                if (item->distance(other) <= delta) {
                        count++;
                }
                other = other->getNext();
        }
        return count;
}
```

Compares <item> to every ListItem in the list with <other> as first element.
Returns the number of points in this list closer than <delta> to <item>

# Solution to HW3

```
void procedure() {
        int i, j;
        int dim = (int) ceil ((double) maxCoord/delta);
        ListItem ***matrix = new ListItem ** [dim];
        for (i = 0; i != dim; i++) {
                matrix[i] = new ListItem * [dim];
                for (j = 0; j != dim; j++) {
                        matrix[i][j] = 0;
                }
        }
        // …
}
```

# Solution to HW3

```
void procedure() {
        // …
        for (i = 0; i != nbPoints; i++) {
                int row = (int)floor(x[i] / delta);
                int col = (int)floor(y[i] / delta);
                ListItem *item = new ListItem(x[i], y[i]);
                ~~if (matrix[row][col] == 0) {~~
                        ~~matrix[row][col] = item;~~
                ~~} else {~~
                        item->setNext(matrix[row][col]);
                        matrix[row][col] = item;
                ~~}~~
        }
        // …
}
```

# Solution to HW3

```
void procedure() {
    // …
    int count = 0;
    for (i = 0; i != dim; i++) {
        for (j = 0; j != dim; j++) {
            ListItem *current = matrix[i][j];
            while (current != 0) {
                if (i+1 < dim) count += checkFrom(current, matrix[i+1][j]);
                if (j+1 < dim) count += checkFrom(current, matrix[i][j+1]);
                if ((i+1 < dim) && (j+1 < dim)) count += checkFrom(current, matrix[i+1][j+1]);
                if ((i+1 < dim) && (j-1 >= 0)) count += checkFrom(current, matrix[i+1][j-1]);
                count += checkFrom(current, current->getNext());
                current = current->getNext();
            }
        }
    }
    // …
}
```

# Solution to HW3

```
void procedure() {
        // …
        printf("Count = %d\n", count);
        // cleanup
        for (i = 0; i != dim; i++) {
                for (j = 0; j != dim; j++) {
                        delete matrix[i][j];
                }
                delete matrix[i];
        }
        delete matrix;
}
```

# HW3: results

| *Name* | Slides? | Correct? | Empty Cell? | Up to Spec? | Leakfree? | n=10,000 | n=25,000 |
|---|---|---|---|---|---|---|---|
| Ari | 1 | 1 | 1 | 1 | 1 | 0,14 | 1,04 |
| Inès | 1 | 1 | 1 | 1 | 1 | 1,00 | 5,50 |
| Qinyan | 1 | 1 | 1 | 1 | 1 | 1,04 | 5,60 |
| Siyi2 | 1 | 1 | 1 | 0 | 1 | 0,03 | 0,37 |
| Hucheng | 1 | 0 | 1 | 1 | 1 | 0,83 | 4,54 |
| Yizheng | 1 | 1 | 1 | 0 | 1 | 1,00 | 5,50 |
| Siyi1 | 1 | 1 | 1 | 0 | 1 | 1,02 | 5,54 |
| Omar | 1 | 1 | 1 | 0 | 1 | 1,95 | 10,76 |