|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Report**  제목: Bin Packing   |  |  |  | | --- | --- | --- | | **<과제물 제출 전 체크리스트>** | | | | 1 | 이 과제물은 내가/우리가 직접 연구하고 작성한 것이다. | O | | 2 | 인용한 모든 자료(책, 논문, 인터넷자료 등)의 인용표시를 바르게 하였다. | O | | 3 | 인용한 자료의 표현이나 내용을 왜곡하지 않았다. | O | | 4 | 정확한 출처 제시 없이 다른 사람의 글이나 아이디어를 가져오지 않았다. | O | | 5 | 정확한 출처 제시 없이 여러 사람의 글이나 아이디어를 짜깁기하지 않았다. | O | | 6 | 과제물 작성 중 도표나 데이터를 조작(위조 또는 변조)하지 않았다. | O | | 7 | 과제물은 다른 사람으로부터 받거나 구매하여 제출하지 않았다. | O | | 8 | 이 과제물에 실질적으로 참여하지 않은 사람을 공동 제출자로 명기하지 않았다. | O | | 9 | 이 과제물과 동일한 내용을 다른 교과목의 과제물로 제출한 적이 없다. | O |  |  |  | | --- | --- | | 과목명 | IMEN281 | | 담당교수 | 김병인 | | 학 과 | 산업경영공학과 | | 학 번 | 20130220 | | 성 명 | 이다진 | | 제출일 | 2017년 12월 8일 | |

목차

**Ⅰ. 목적1**

**Ⅱ. 방법과 알고리즘1**

1. 전체 구조1

2. 알고리즘1

3. 클래스 구성5

**Ⅲ. 수행 결과에 대한 토의7**

1. 토론7

2. 결론7

**Ⅳ. 실행 화면 스크린샷8**

Ⅰ. 목적

본 프로그램은 C++로 텍스트 파일의 데이터를 읽고 알맞은 클래스에 저장한 후 휴리스틱 알고리즘을 이용하여 Bin Packing 문제의 솔루션을 구한다. 프로그램 구현을 통해 C++을 통한 파일 입출력, 클래스 구성, STL의 활용법을 익히고 Bin Packing 알고리즘을 이해한다. Console(Mac OS)에서 프로그램 실행을 통해 Text User Interface 환경에 익숙해진다. 이러한 과정을 통해 컴퓨터로 정보를 처리하고 문제의 솔루션을 찾는 기술을 익힌다.

Ⅱ. 방법과 알고리즘

1. 전체 구조

(1) 명령 인자로 아이템 정보가 담긴 파일의 이름, Bin Size, 실행할 알고리즘 타입, 결과를 저장할 파일의 이름을 차례대로 받는다.

(2) 아이템 정보가 담긴 파일로부터 아이템 객체를 생성하고 벡터 컨테이너에 담는다.

(3) 입력 받은 타입의 알고리즘으로 Bin Packing 문제의 솔루션을 찾는다.

(4) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

(5) 메인 프로그램을 종료한다.

2. 알고리즘

본 알고리즘에서 Truck은 Bin을 의미한다.

(1) 아이템 정보가 담긴 파일로부터 아이템 객체를 생성하고 벡터 컨테이너에 담는다.

1) 아이템 파일을 오픈하고 실패 시 예외처리하고 메소드를 종료한다.

2) 첫번째 줄을 제외한 나머지 줄에 대하여 ‘\t’을 구분자로 아이템의 id, weight를 저장한다.

3) 저장한 값으로 아이템 객체를 생성하고 벡터 컨테이너 items에 담는다.

4) 파일의 끝에 도달하면 종료한다.

(2) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

1) 결과를 출력할 파일을 오픈하고 실패 시 예외처리하고 메소드를 종료한다.

2) 정해진 형식대로 Input file 명, Bin 용량, Algorithm Type, Bin 수, 계산시간을 출력한다.

3) 모든 Bin에 대해 차례대로 Bin 안의 item 정보를 출력한다.

(3) First Fit 알고리즘

1) clock() 함수로 시작 시간을 기록한다.

2) weight가 Bin Size(=Truck Capacity)보다 큰 아이템을 items에서 제거한다. 모든 아이템이 제거되었을 경우 ‘e’를 throw하고 메소드를 종료한다.

items의 모든 아이템에 대해 순서대로

3-1) 현재 아이템의 무게가 첫번째 트럭의 남은 용량보다 작거나 같으면 트럭에 넣고 트럭의 남은 용량을 조절한다. 현재 트럭의 남은 용량보다 크면 다음 트럭에 대해 검사한다.

3-2) 모든 트럭에 대해 아이템을 넣지 못할 경우 trucks에 새로운 트럭을 생성하여 추가하고 새로운 트럭에 아이템을 넣는다.

4) clock() 함수로 종료 시간을 기록하여 run time을 계산한다.

5) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

(4) First Fit Decreasing 알고리즘

1) clock() 함수로 시작 시간을 기록한다.

2) weight가 Bin Size(=Truck Capacity)보다 큰 아이템을 items에서 제거한다. 모든 아이템이 제거되었을 경우 ‘e’를 throw하고 메소드를 종료한다.

3) STL의 sort함수를 이용하여 items의 아이템을 무게를 기준으로 내림차순으로 정렬한다.

items의 모든 아이템에 대해 순서대로

4-1) 현재 아이템의 무게가 첫번째 트럭의 남은 용량보다 작거나 같으면 트럭에 넣고 트럭의 남은 용량을 조절한다. 현재 트럭의 남은 용량보다 크면 다음 트럭에 대해 검사한다.

4-2) 모든 트럭에 대해 아이템을 넣지 못할 경우 trucks에 새로운 트럭을 생성하여 추가하고 새로운 트럭에 아이템을 넣는다.

5) clock() 함수로 종료 시간을 기록하여 run time을 계산한다.

6) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

(5) Best Fit 알고리즘

1) clock() 함수로 시작 시간을 기록한다.

2) weight가 Bin Size(=Truck Capacity)보다 큰 아이템을 items에서 제거한다. 모든 아이템이 제거되었을 경우 ‘e’를 throw하고 메소드를 종료한다.

items의 모든 아이템에 대해 순서대로

3-1) trucks의 트럭을 남은 용량을 기준으로 오름차순으로 정렬한다.

3-2) 현재 아이템의 무게가 첫번째 트럭(=남은 용량이 가장 작은 트럭)의 남은 용량보다 작거나 같으면 트럭에 넣고 트럭의 남은 용량을 조절한다. 현재 트럭의 남은 용량보다 크면 다음 트럭에 대해 검사한다.

3-3) 모든 트럭에 대해 아이템을 넣지 못할 경우 trucks에 새로운 트럭을 생성하여 추가하고 새로운 트럭에 아이템을 넣는다.

4) clock() 함수로 종료 시간을 기록하여 run time을 계산한다.

5) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

(6) Best Fit Decreasing 알고리즘

1) clock() 함수로 시작 시간을 기록한다.

2) weight가 Bin Size(=Truck Capacity)보다 큰 아이템을 items에서 제거한다. 모든 아이템이 제거되었을 경우 ‘e’를 throw하고 메소드를 종료한다.

3) STL의 sort함수를 이용하여 items의 아이템을 무게를 기준으로 내림차순으로 정렬한다.

items의 모든 아이템에 대해 순서대로

4-1) trucks의 트럭을 남은 용량을 기준으로 오름차순으로 정렬한다.

4-2) 현재 아이템의 무게가 첫번째 트럭(=남은 용량이 가장 작은 트럭)의 남은 용량보다 작거나 같으면 트럭에 넣고 트럭의 남은 용량을 조절한다. 현재 트럭의 남은 용량보다 크면 다음 트럭에 대해 검사한다.

4-3) 모든 트럭에 대해 아이템을 넣지 못할 경우 trucks에 새로운 트럭을 생성하여 추가하고 새로운 트럭에 아이템을 넣는다.

5) clock() 함수로 종료 시간을 기록하여 run time을 계산한다.

6) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

(7) First Fit Decreasing-Last Two Fit 알고리즘

1) clock() 함수로 시작 시간을 기록한다.

2) weight가 Bin Size(=Truck Capacity)보다 큰 아이템을 items에서 제거한다. 모든 아이템이 제거되었을 경우 ‘e’를 throw하고 메소드를 종료한다.

3) STL의 sort함수를 이용하여 items의 아이템을 무게를 기준으로 내림차순으로 정렬한다.

items의 모든 아이템에 대해 순서대로

4-1) 현재 아이템의 무게가 items의 마지막 두 아이템 무게의 합보다 작은 경우, 마지막 두 아이템을 선택한다.

4-2) 마지막 두 아이템 무게의 합이 첫번째 트럭의 남은 용량보다 작거나 같으면 트럭에 넣고 트럭의 남은 용량을 조절한다. 현재 트럭의 남은 용량보다 크면 다음 트럭에 대해 검사한다.

4-3) 모든 트럭에 대해 아이템을 넣지 못할 경우 trucks에 새로운 트럭을 생성하여 추가하고 새로운 트럭에 아이템을 넣는다.

5-1) 현재 아이템의 무게가 items의 마지막 두 아이템 무게의 합보다 큰 경우, 현재 아이템을 선택한다.

5-2) 현재 아이템의 무게가 첫번째 트럭의 남은 용량보다 작거나 같으면 트럭에 넣고 트럭의 남은 용량을 조절한다. 현재 트럭의 남은 용량보다 크면 다음 트럭에 대해 검사한다.

5-3) 모든 트럭에 대해 아이템을 넣지 못할 경우 trucks에 새로운 트럭을 생성하여 추가하고 새로운 트럭에 아이템을 넣는다.

6) clock() 함수로 종료 시간을 기록하여 run time을 계산한다.

7) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

(8) Iterative Knapsack 알고리즘

1) 아이템 객체를 생성할 때 profit 변수에 weight와 같은 값을 저장한다.

2) clock() 함수로 시작 시간을 기록한다.

3) weight가 Bin Size(=Truck Capacity)보다 큰 아이템을 items에서 제거한다. 모든 아이템이 제거되었을 경우 ‘e’를 throw하고 메소드를 종료한다.

4-1) items의 모든 아이템을 Knapsack 알고리즘으로 첫번째 트럭에 넣고 items에서 제거한다.

4-2) items에 아이템이 남지 않을 때 까지 새로운 트럭을 생성하여 4-1을 실행한다.

5) clock() 함수로 종료 시간을 기록하여 run time을 계산한다.

6) 정해진 형식대로 텍스트 파일에 결과를 출력한다.

3. 클래스 구성

(1) class Item

class Item {

public:

    int id; // id of the item

    int w;  // weight of the item

    int p;  // profit of the item

    Item(int \_id, int \_w) : id(\_id), w(\_w) {}    // constructor

    Item(int \_id, int \_w, int \_p) : id(\_id), w(\_w), p(\_p) {}

// constructor for knapsack algorithm

    bool operator >(const Item &a) const {  // operator overloading

        return this->w > a.w;

    }

};

(2) class Truck

class Truck {

public:

    vector<Item> innerItems;   // inner items of the truck

    int c;                      // capacity of the truck

    Truck(int \_c) : c(\_c) {}   // constructor

    bool operator <(const Truck &a) const {  // operator overloading

        return this->c < a.c;

    }

};

(3) class BinPacking

class BinPacking {

public:

    // members

    vector<Item> items;    // vector container of items

    vector<Truck> trucks;  // vector container of trucks

    int capacity;        // capacity of the truck

    string inputfileName;

    string outputfileName;

    string algorithmName;

    // method reaing input file

    void readInputfile(string inputfile, string outputfile, int c);

    // method writing solution file

    void writeSolutionfile(string filename, double runTime);

    // algorithms

    void firstFit();

    void firstFitDecreasing();

    void bestFit();

    void bestFitDecreasing();

    void firstFitDecreasing\_lastTwoFit();

};

(4) class MyKnapsack

class Myknabsack {

public:

    // members

    vector<Item> items; // vector container of items

    int capacity; // vector container of trucks

    vector<Truck> trucks; // capacity of the truck

    vector<int> weight; // vector container of weights of items

    string inputfileName;

    string outputfileName;

    string algorithmName;

    // algorithm

    void solve\_iterative();

    // method reaing input file

    void readInputfile(string inputfile, string outputfile, int c);

    // method writing solution file

    void writeSolutionfile(string filename, double runTime);

};

Ⅲ. 수행 결과에 대한 토의

1. 토론

프로그램 개발 과정 중 발생한 문제와 해결 방법에 대해 서술한다.

(1) 트럭의 용량보다 무거운 아이템 처리 문제

트럭의 용량보다 무거운 아이템을 미리 제거하여 빈 트럭이 생기지 않도록 하였다.

(2) Mac Xcode에서 작성한 코드를 Console에서 실행할 때 발생한 문제

       // 수정 전 코드

vector<Item>::iterator i = items.begin();

       for (i; i != items.end(); i++) {

           ...

}

       // 수정 후 코드

vector<Item>::iterator i = items.begin();

        while (true) {

            if (i->w > capacity)

                items.erase(i);

            else

                i++;

            if (i == items.end())

                break;

        }

items의 모든 아이템을 검사하여 특정 아이템을 삭제하기 위해 for문을 사용하였다. 이때 Xcode에서는 수정 전 코드에 문제가 없었지만 Console에서는 반복자가 가리키는 값이 명확하지 않다는 오류가 발생하였다. erase() 함수 실행 시 반복자를 증가시키지 않아도 자동으로 증가하여 발생하는 문제로, while문을 이용하여 직접 반복자를 관리할 수 있도록 수정하였다.

2. 결론

본 프로그램 작성을 통해 C++의 생성자, 연산자 오버로딩, File I/O, STL 사용법을 익히는 것뿐만 아니라 다양한 알고리즘을 이해하고 코드화하는 연습을 했다. 또한 최적해는 아니지만 사람의 상식적인 사고로 최적해에 근접하는 솔루션을 도출하는 휴리스틱 알고리즘을 공부하고 구현했다. 그리고 김병인 교수님께서 개발하신 First Fit Decreasing-Last Two Fit을 접해봄으로써 기존의 유명한 알고리즘을 수동적으로 이용하는데 그치지 않고, 더욱 최적해에 가까운 솔루션을 도출하기 위해 이를 개선하고 활용하는 자세를 배울 수 있었다.

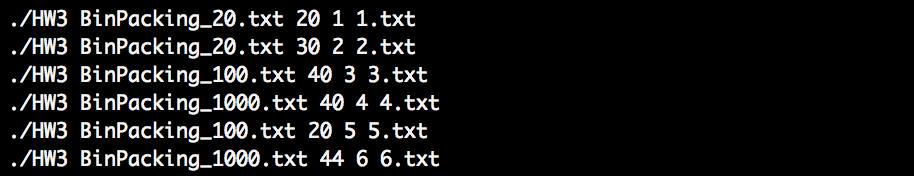
과제에 소요된 시간 총 6시간

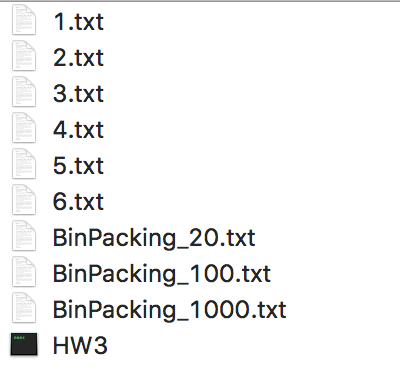
알고리즘 공부 1시간 + 코드 작성 및 수정 2시간 30분

+ Console실행 시 발생한 문제 수정 1시간 + 보고서 작성 1시간 30분

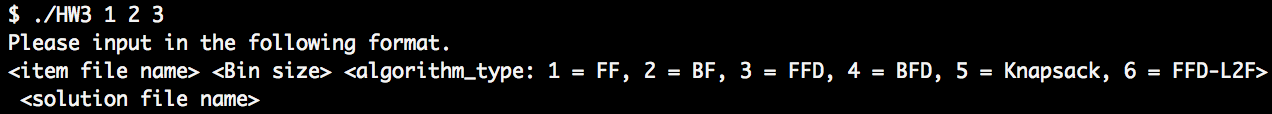
Ⅳ. 실행 화면 스크린샷

1. 실행 성공





2-1. 잘못된 명령 인자 전달 경우(인자의 수가 맞지 않음)



2-2. 잘못된 명령 인자 전달 경우(해당하는 알고리즘 타입이 없음)



3. 아이템을 한 개도 담을 수 없는 경우

