問題1−1

・問題文

始めに商品情報のデータ型 SHOHIN を下記のように定める。

```
#define ZEI (1.08)
typedef struct{
  char* name;
  int tanka;
  int sotozei;
} SHOHIN;
```

name は商品名を表す文字列、tanka は単価、sotozei は 0 なら単価は消費税を含むが、 1 なら外税表示のため実際の販売単価は tanka*ZEI となることを意味す る。

これを表示する `void printshohin(SHOHIN p)` を作成せよ。 但し、この関数で表示するための printf に制御文字列は "%s\t単価%d円(%s)" とし、例えば次のように表示させるとする(改行しない)。

```
Apple      単価150円(内税)
```

shohin.h, shohin.c を下記のように定める。配列shohin は商品リストを表 し、単価が 0 の項目を番兵とする。

・ソースコード

```
#include <stdio.h>
#define ZEI (1.08)

typedef struct
{
    char *name;
    int tanka;
    int sotozei;
} SHOHIN;

SHOHIN shohin[] = {{"Apple", 150}, {"Orange", 100}, {"Banana",
200}, {"Book1", 500, 1}, {"", 0}};

void printshohin(SHOHIN s)
{

    char *uti = "内税";
    char *soto = "外税";
    char *name = s.name;
    int sotozei = s.sotozei;
    int tanka = s.tanka;

    if (s.sotozei == 0)
    {
        printf("%s\t単価%d円(%s)", name, tanka, uti);
    }
    else
    {
        printf("%s\t単価%d円(外税)", name, tanka, soto);
    }
}
int main(void)
{
    int i;
    for (i = 0; shohin[i].tanka != 0; i++)
    {
        printf("商品コード %d 品名 ", i);
        printshohin(shohin[i]);
```

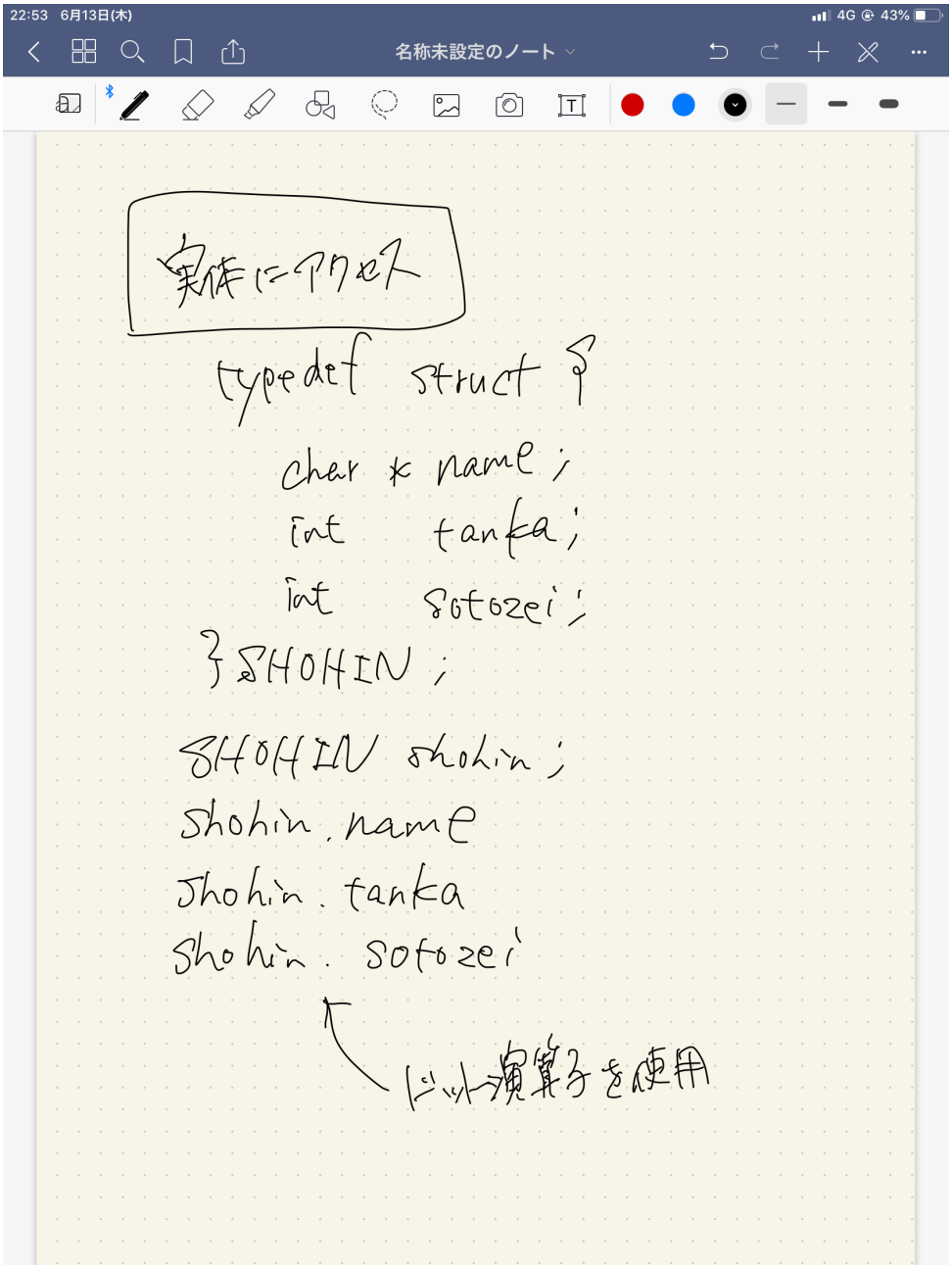
```
        printf("\n");
    }
    return 0;
}
```

・実行結果

```
10kaoru12@ishizawakaorunoMacBook-Pro ~/B/G/2/R/1>
cd "/Users/10kaoru12/Box Sync/GitHub/2y_university_prog
ramming_report/Report_Task_1/1/" && gcc shohin.c -o sho
hin && "/Users/10kaoru12/Box Sync/GitHub/2y_university_
programming_report/Report_Task_1/1/"shohin
商品コード 0 品名 Apple 単価150円(内税)
商品コード 1 品名 Orange 単価100円(内税)
商品コード 2 品名 Banana 単価200円(内税)
商品コード 3 品名 Book1 単価500円(外税)
```

・説明

この問題では、問題設定に制御文字の制限があるため、char型で内税と外税を別途変数で記述する必要がある。
if文でSHOHIN型の構造体配列shohinメンバであるsotozeiにアクセスし、外税であるか内税であるかを判断している。
今回の問題では商品名、単価を表示すれば良いので、main関数からSHOHIN型の構造体であるshohinを構造体変数sとして実体を受け取り、ドット演算子を用いてsにアクセスする。商品名へのアクセスではs.name、単価へのアクセスではs.tankaとしてアクセスする。



問題1−2

・問題文

売上を示す構造体を次に示す。

```
typedef struct {
  int code;
  int num;
}URIAGE;
```

code は配列 shohin の添字、num は売上個数を示す。

この時、この構造体のポインターを受け取り、商品名、単価、内税／外税の 区別、個数、税込み購入価格を表示し、税込み購入価格(int型)を返す `int printUriage(URIAGE* q)`関数を作りなさい。 そして、下記のプログラムと結合し、実行結果を報告せよ。

・ソースコード

```
#include <stdio.h>
#define ZEI 1.08

typedef struct
{
    char *name;
    int tanka;
    int sotozei;
} SHOHIN;

typedef struct
{
    int code;
    int num;
} URIAGE;

SHOHIN shohin[] = {{"Apple", 150}, {"Orange", 100}, {"Banana",
200}, {"Book1", 500, 1}, {"", 0}};

int printUriage(URIAGE *q)
{
    int sotozei = shohin[q->code].sotozei;
    int tanka = shohin[q->code].tanka;
    int num = q->num;
    int syoukei = shohin[q->code].tanka * q->num;
    char *name = shohin[q->code].name;
    double zeisyoukei = shohin[q->code].tanka * q->num * ZEI;

    if (sotozei == 0)
    {

        printf("%s\t単価%d円(内税) %d 個\t%d 円\n", name, tanka, num,
syoukei);
        return syoukei;
    }
    else
    {
```

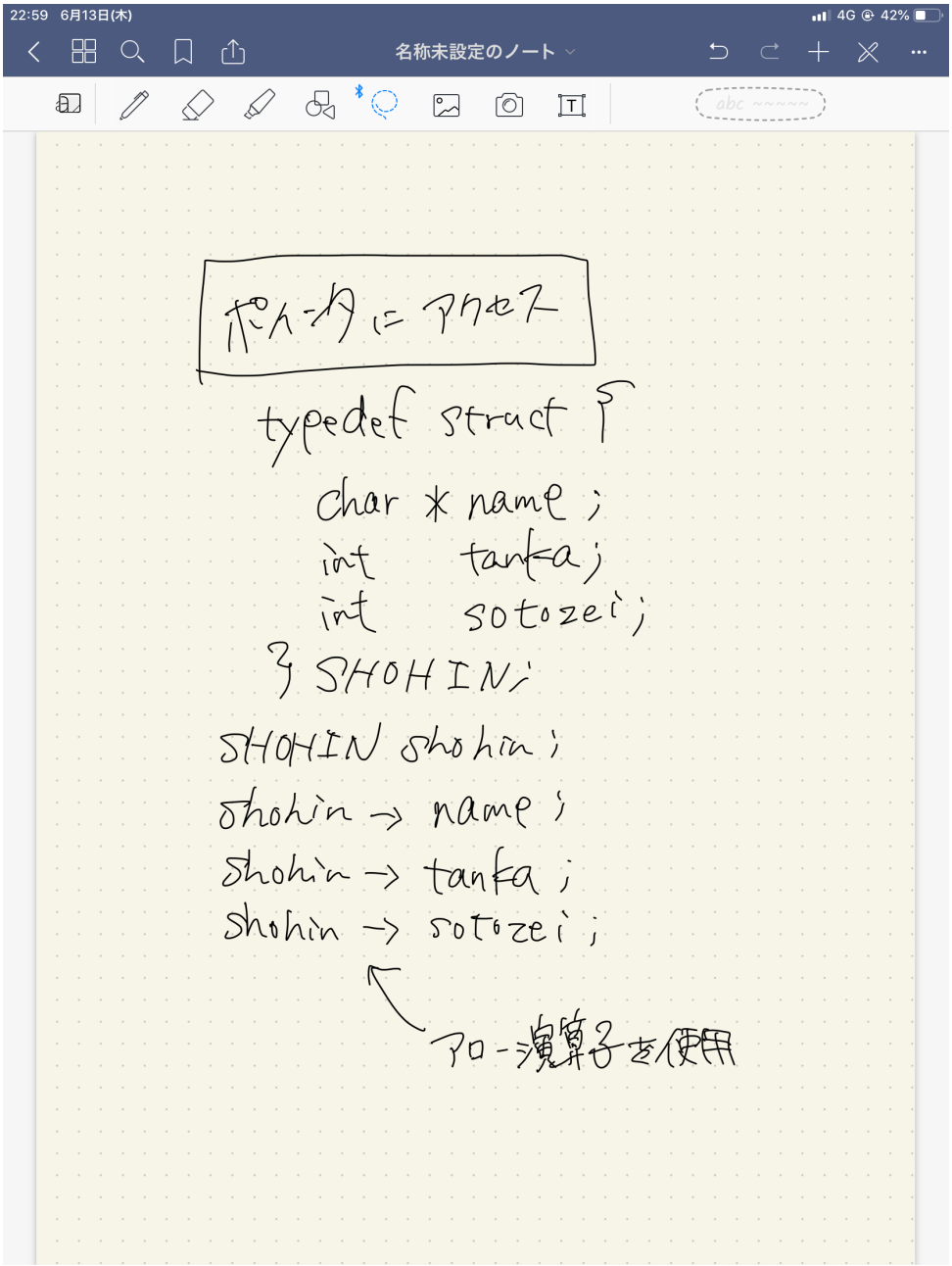
```
        printf("%s\t単価%d円(外税) %d 個\t%.0f 円\n", name, tanka,
num, zeisyoukei);
        return zeisyoukei;
    }
}
int main(void)
{
    URIAGE u1 = {0, 3};
    URIAGE u2 = {3, 4};
    int shokei;
    shokei = printUriage(&u1);
    printf("\t%d\n", shokei);
    shokei = printUriage(&u2);
    printf("\t%d\n", shokei);
    return 0;
}
```

・実行結果

```
10kaoru12@ishizawakaorunoMacBook-Pro ~/B/G/2/R/2>
cd "/Users/10kaoru12/Box Sync/GitHub/2y_university_programmi
ng_report/Report_Task_1/2/" && gcc uriage.c -o uriage && "/U
sers/10kaoru12/Box Sync/GitHub/2y_university_programming_rep
ort/Report_Task_1/2/"uriage
Apple   単価150円(内税) 3 個    450 円
        450
Book1   単価500円(外税) 4 個    2160 円
        2160
```

・説明

この問題では、URIAGE型の構造体u1,u2をポインタとしてポインタとして受け取るので、アロー演算子を使用してu1,u2で指定された商品コードと商品数にアクセスする。
if文を使って外税と内税の判別をする。
その際、SHOHINは構造体配列であるので配列参照演算子とアロー演算子を用いてshohin[q->code].nameなどとアクセスする。
個数のアクセスでは、main関数からポインタとして受け取ったu1,u2をqとしてq->numでアクセスする。
税込価格の表示では、あらかじめ定義されたZEIを用いて、shohin[q->code].tanka*q->num*ZEIで計算する。
この際に注意すべきなのはprintfに直接入れる場合、ZEIがfloatに値するので%fにしないとオーバーフローが起きる.



問題1−3

・問題文

番兵としてcode=−1を持つURIAGE の配列を渡すと、各売上を表示し、最後に 小計を出力し、さらにその小計を返す `int printUriageArray(URIAGE u[])` を作りなさい。そして、下記のプログラムと結合し、実行結果を報告せよ。

・ソースコード

```
#include <stdio.h>
#define ZEI 1.08

typedef struct
{
    char *name;
    int tanka;
    int sotozei;
} SHOHIN;

typedef struct
{
    int code;
    int num;
} URIAGE;

SHOHIN shohin[] = {{"Apple", 150}, {"Orange", 100}, {"Banana",
200}, {"Book1", 500, 1}, {"", 0}};

int printUriageArray(URIAGE u[])
{
    int i;
    int sotozei;
    int tanka;
    int num;
    int syoukei;
    char *name;
    double zeisyoukei;
    double sum;
    for (i = 0; u[i].code != (-1); i++)
    {

        name = shohin[u[i].code].name;
        sotozei = shohin[u[i].code].sotozei;
        tanka = shohin[u[i].code].tanka;
        num = u[i].num;
        syoukei = shohin[u[i].code].tanka * u[i].num;
```

```
        zeisyoukei = shohin[u[i].code].tanka * u[i].num * ZEI;

        if (sotozei == 0)
        {
            printf("%s\t単価%d円(内税) %d 個\t%d 円\n", name, tanka,
num, syoukei);
            sum += syoukei;
        }
        else
        {

            printf("%s\t単価%d円(外税) %d 個\t%.0f 円\n", name,
tanka, num, zeisyoukei);
            sum += zeisyoukei;
        }
    }
    printf("小計:\t\t\t\t%.0f円\n", sum);
    return sum;
}
int main(void)
{
    URIAGE u[] = {{0, 3}, {1, 2}, {2, 1}, {3, 4}, {-1}};
    int shokei;
    shokei = printUriageArray(u);
    printf("\t%d\n", shokei);
    return 0;
}
```

・実行結果

```
10kaoru12@ishizawakaorunoMacBook-Pro ~/B/G/2/R/2>
cd "/Users/10kaoru12/Box Sync/GitHub/2y_university_programmi
ng_report/Report_Task_1/3/" && gcc shokei.c -o shokei && "/U
sers/10kaoru12/Box Sync/GitHub/2y_university_programming_rep
ort/Report_Task_1/3/"shokei
Apple   単価150円(内税) 3 個    450 円
Orange  単価100円(内税) 2 個    200 円
Banana  単価200円(内税) 1 個    200 円
Book1   単価500円(外税) 4 個    2160 円
小計:                          3010円
        3010
```

・説明

この問題では、構造体配列uを実体受け取り順番にアクセスしてuにある商品コードと商品数を小計するプログラムである。
今回構造体配列uは番兵として-1が指定されているのでfor文でu[i].code!=(-1)を継続条件としてループを回す。
その中でif文で外税か内税かの判断を行う、この問題ではuが実体として渡されているので、ドット演算子でアクセスする。
この際、uは構造体配列なので、uは配列参照演算子をfor文のイテレータを使用してアクセスする必要がある。
例としては、shohin[u[i].code].sotozeiなどである。
ループで順々にshohinの配列参照演算子の中でURIAGE型のメンバであるcodeにアクセスし、対象のshohinを特定し、ドット演算子で商品名、単価にアクセスして、単価とuのメンバであるnumと消費税であるZEIを掛けて内税を計算、外税の場合は消費税はかけずに計算する。
そして、numと消費税と単価をかけて出た値をループで足していき小計を求める。

問題1−4

・問題文

番兵として NULL を持つ、売上の配列の番地の配列を受け取り、 各売上配列ごとに売上の表示と小計を出力し、----------で区切っ て表示し、 最後に合計金額を表示し、合計金額を返す `int printUriageTrans(URIAGE** u)` を作りなさい。そして、下記のプログラムと結合し、実行結果を報告せよ。

```
#include <stdio.h>
#include "shohin.h"
#include "uriage.h"
int main(void){
  URIAGE uriage0[]={{2,1},{1,6},{-1}};
  URIAGE uriage1[]={{0,3},{1,2},{2,1},{-1}};
  URIAGE uriage2[]={{3,1},{-1}};
  URIAGE uriage3[]={{1,3},{0,1},{-1}};
  URIAGE uriage4[]={{1,3},{0,1},{3,2},{-1}};
  URIAGE* uriage[]={uriage0, uriage1, uriage2, uriage3, uriage4,
NULL};
  int total;
  total=printUriageTrans(uriage);
  printf("%d\n",total);
  return 0;
}
```

・ソースコード

```
#include <stdio.h>
#define ZEI 1.08

typedef struct
{
    char *name;
    int tanka;
    int sotozei;
} SHOHIN;

typedef struct
{
    int code;
    int num;
} URIAGE;

SHOHIN shohin[] = {{"Apple", 150}, {"Orange", 100}, {"Banana",
200}, {"Book1", 500, 1}, {"", 0}};

int printUriageTrans(URIAGE **u)
{
    int i, j;
    double sum = 0, all = 0;
    for (i = 0; u[i] != NULL; i++)
    {
        for (int j = 0; u[i][j].code != (-1); j++)
        {
            int sotozei = shohin[u[i][j].code].sotozei;
            char *name = shohin[u[i][j].code].name;
            int tanka = shohin[u[i][j].code].tanka;
            int num = u[i][j].num;
            int syoukei = shohin[u[i][j].code].tanka * u[i]
[j].num;
            double zeisyoukei = shohin[u[i][j].code].tanka * u[i]
[j].num * ZEI;

            if (sotozei == 0)
            {
```

```
                printf("%s\t単価%d円(内税) %d 個\t%d 円\n", name,
tanka, num, syoukei);
                sum += syoukei;
            }
            else
            {
                printf("%s\t単価%d円(外税) %d 個\t%.0f 円\n", name,
tanka, num, zeisyoukei);
                sum += zeisyoukei;
            }
        }
        printf("小計:\t\t\t\t%.0f円\n", sum);
        printf("------------------\n");
        all += sum;
        sum = 0;
    }
    printf("合計:\t\t\t\t%.0f円\n", all);
    return all;
}

int main(void)
{
    URIAGE uriage0[] = {{2, 1}, {1, 6}, {-1}};
    URIAGE uriage1[] = {{0, 3}, {1, 2}, {2, 1}, {-1}};
    URIAGE uriage2[] = {{3, 1}, {-1}};
    URIAGE uriage3[] = {{1, 3}, {0, 1}, {-1}};
    URIAGE uriage4[] = {{1, 3}, {0, 1}, {3, 2}, {-1}};
    URIAGE *uriage[] = {uriage0, uriage1, uriage2, uriage3,
uriage4, NULL};
    int total;
    total = printUriageTrans(uriage);
    printf("%d\n", total);
    return 0;
}
```

・実行結果

```
10kaoru12@dot1x7553 ~/B/G/2/R/4> cd "/Users/10kaoru12/Box Sync
/GitHub/2y_university_programming_report/Report_Task_1/4/" &&
gcc uriagetrans.c -o uriagetrans && "/Users/10kaoru12/Box Sync
/GitHub/2y_university_programming_report/Report_Task_1/4/"uria
getrans
Banana   単価200円(内税) 1 個     200 円
Orange   単価100円(内税) 6 個     600 円
小計:                             800円
------------------
Apple    単価150円(内税) 3 個     450 円
Orange   単価100円(内税) 2 個     200 円
Banana   単価200円(内税) 1 個     200 円
小計:                             850円
------------------
Book1    単価500円(外税) 1 個     540 円
小計:                             540円
------------------
Orange   単価100円(内税) 3 個     300 円
Apple    単価150円(内税) 1 個     150 円
小計:                             450円
------------------
Orange   単価100円(内税) 3 個     300 円
Apple    単価150円(内税) 1 個     150 円
Book1    単価500円(外税) 2 個     1080 円
小計:                             1530円
------------------
合計:                             4170円
4170
```

・説明

この問題では、URIAGE型の構造体配列uriage0~5が入ったURIAGE型の構造体配列uriageに順々にアクセスし、uriage0から5までそれぞれの小計を求め、最終的にuriage0から5までの小計を合計したものを返す関数を実装する。
この場合、uriageの番兵はNULLとして設定されているのでループでNULLまでループさせ、その中でuriage0~5までをループさせ、その中で配列参照演算子を二回使用してshohinのメンバにアクセスする。
例として、shohinのsotozeiにアクセスする場合を考えると、
shohin[u[i][j].code].sotozei
とするとアクセスすることができるので、これを使ってuriage0から5までそれぞれの売り上げを小計してそれらを合計したものを出力する。

考察

・問題文

ダイクストラ法をC言語で実装し、グラフ理論の基礎及び最短経路問題への関心と理解を深める。
その際、C言語標準ライブラリ以外使用しないものとし、ライブラリに依存しない実装力の向上を図る。

・ソースコード

```
#include <stdio.h>

#define inf 9999999
#define size 10000

int n, i, j, neigh;
int dist[size][size];
int cost[size];
int flag[size];
int use[size];

int dijkstra(const int start, const int goal)
{
    int min, target;
    cost[start] = 0;
    while (1)
    {
        min = inf;
        for (i = 0; i < n; ++i)
        {
            if (!use[i] && min > cost[i])
            {
                min = cost[i];
                target = i;
            }
        }

        if (target == goal)
        {
            return cost[goal];
        }
        for (neigh = 0; neigh < n; ++neigh)
        {
            if (cost[neigh] > dist[target][neigh] + cost[target])
            {
                cost[neigh] = dist[target][neigh] + cost[target];
            }
        }
```

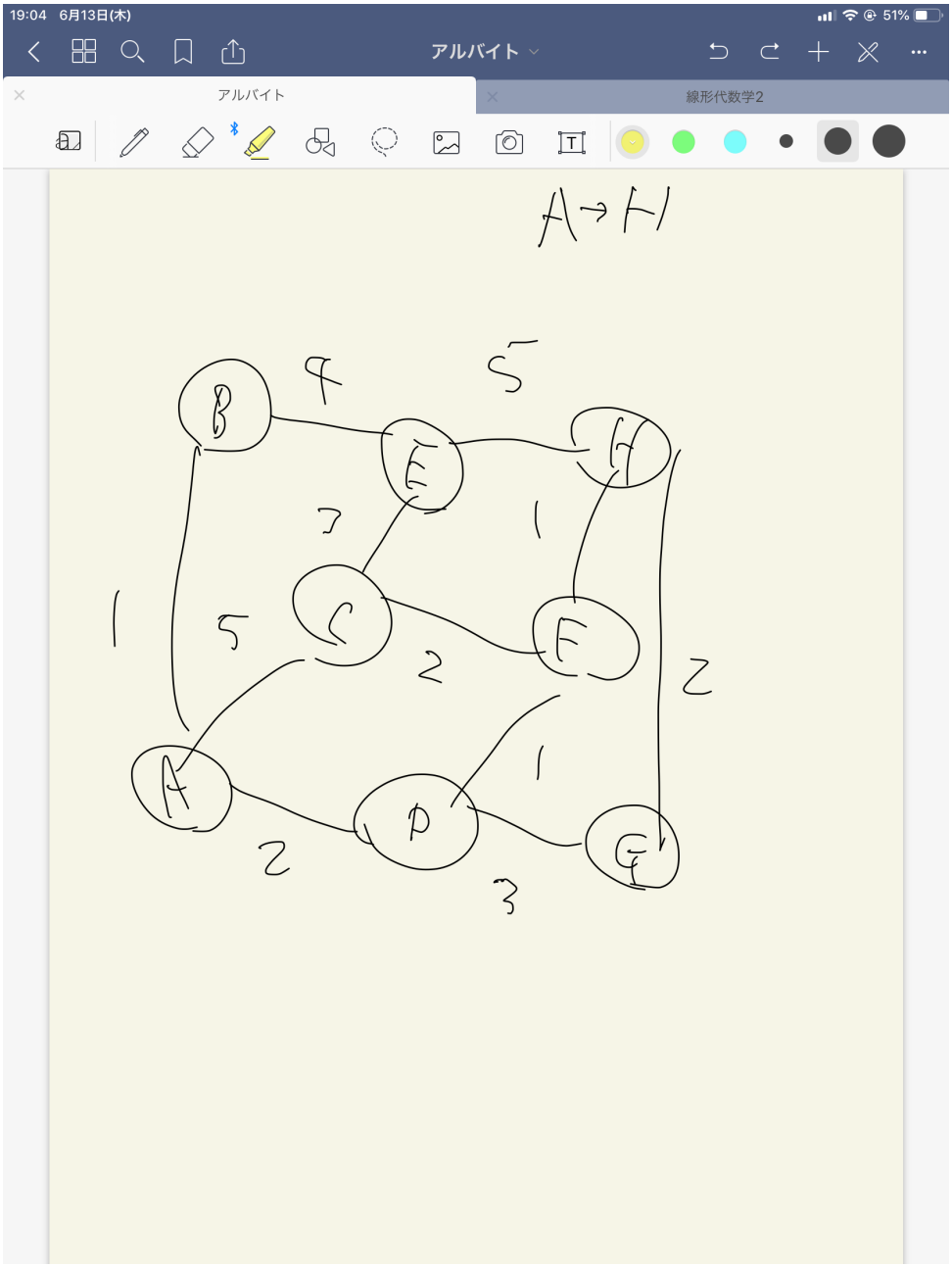
```
        use[target] = 1;
    }
}

int main(void)
{
    int a;
    int b;
    int l;

    int r;
    const int s, g;

    for (i = 0; i < size; ++i)
    {
        cost[i] = inf;
        flag[i] = 0;
        for (j = 0; j < size; ++j)
        {
            dist[i][j] = inf;
        }
    }

    scanf("%d%d", &n, &r);
    for (i = 0; i < r; ++i)
    {
        scanf("%d%d%d", &a, &b, &l);
        dist[a][b] = l;
    }
    scanf("%d%d", &s, &g);
    {
        printf("distance:%d\n", dijkstra(s, g));
    }
    return 0;
}
```

・実行結果

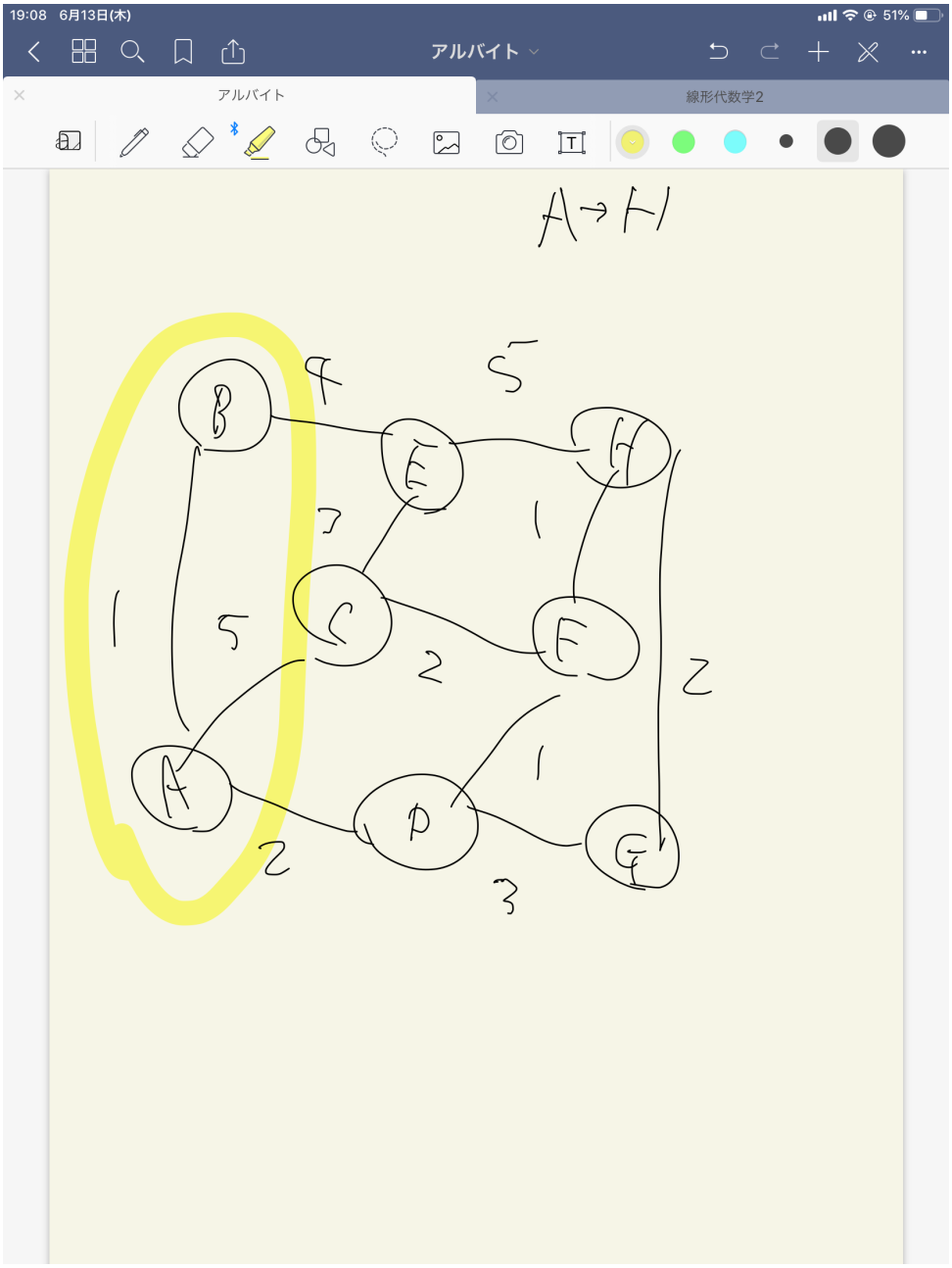
```
10kaoru12@dot1x7553 ~/B/G/2/R/consideration> cd "/Users/10kaoru12/
Box Sync/GitHub/2y_university_programming_report/Report_Task_1/con
sideration/" && gcc consideration.c -o consideration && "/Users/10
kaoru12/Box Sync/GitHub/2y_university_programming_report/Report_Ta
sk_1/consideration/"consideration
7 10
0 1 30
0 3 10
0 2 15
1 3 25
1 4 60
2 3 40
2 5 20
3 6 35
4 6 20
5 6 30
0 6
distance:45
```

・説明

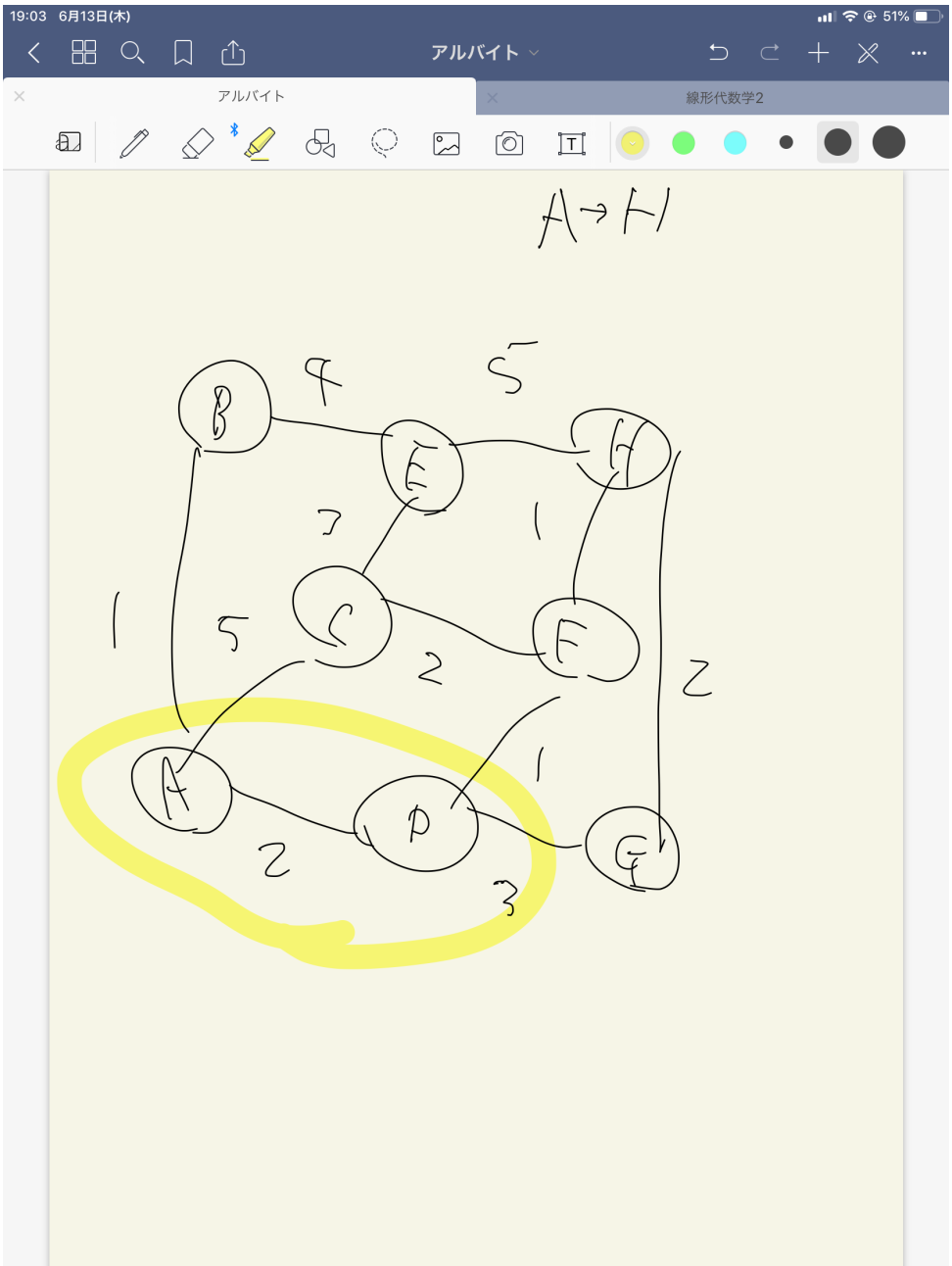
ダイクストラ法では、greedlyにグラフの最適なルートを求めることをしている。
このようなグラフがあり、AからHまでの最短経路を求めるとすると、まず、現在行くことのできるポイントはA→B,A→C,A→Dである。



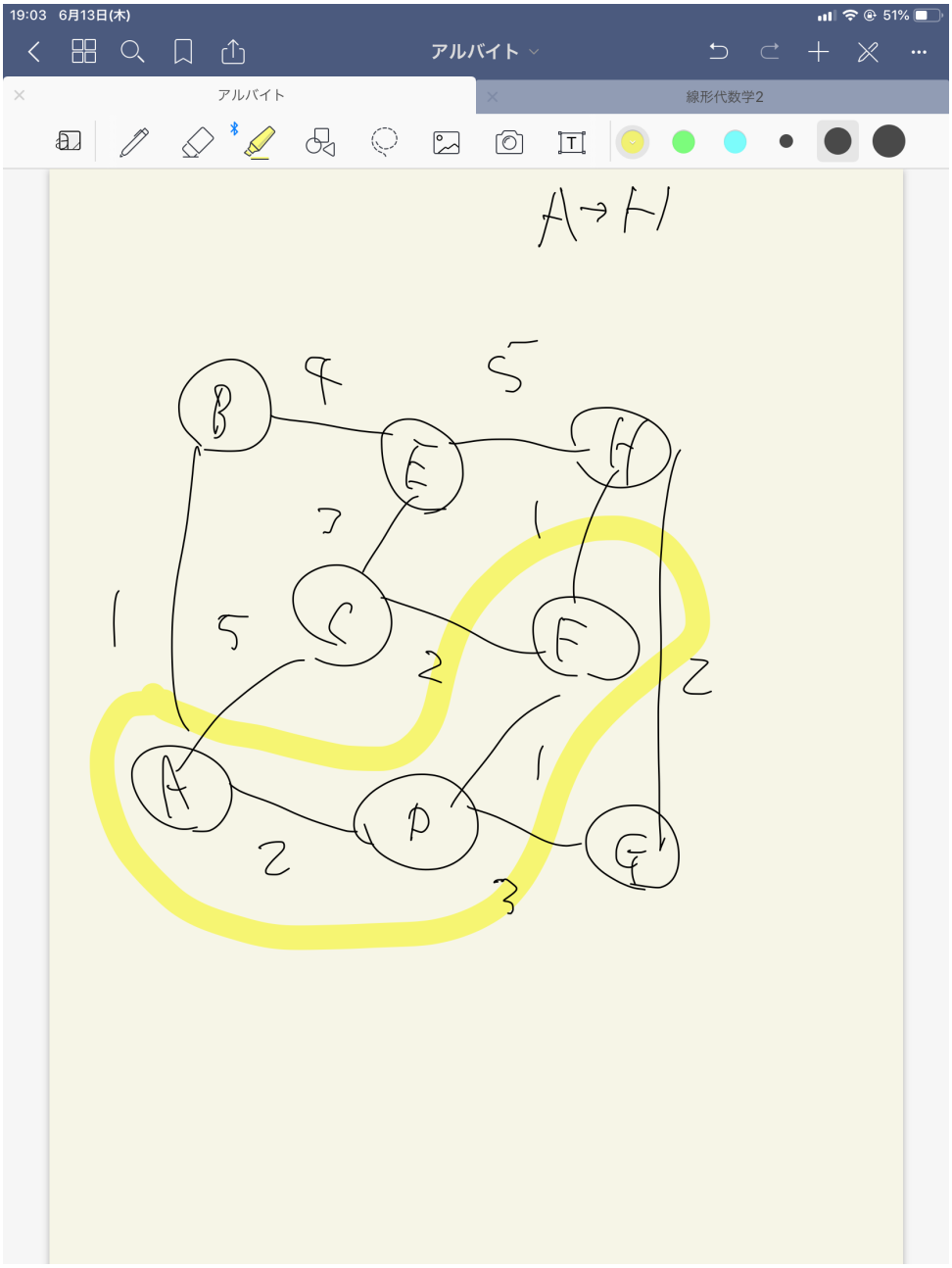
この中で最小のコストでいけるポイントはA→Bであることは下記のグラフを見れば明らかである。
次に、現在いけるポイントはA→B→E,A→C,A→Dである、この中で最小でいけるのはコストが2でいけるA→Dである。



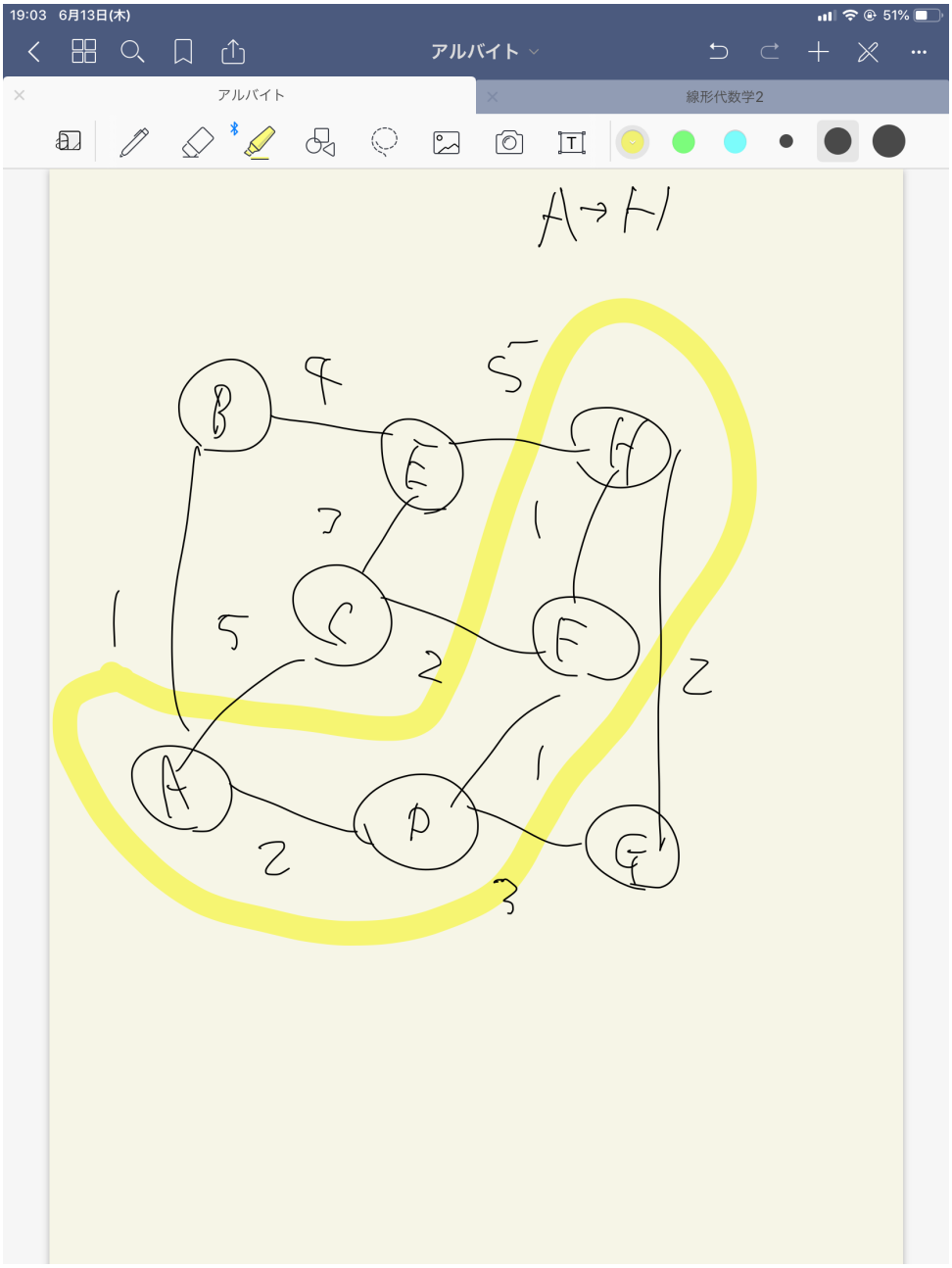
次にいけるポイントはA→B→E,A→C,A→D→F,A→D→Gである。
この中で最小のコストでいけるのはコストが3でいけるA→D→Fであるので、このルートを選ぶ。



次にいけるポイントは、A→B→E,A→C,A→D→F,A→D→F→Hである。
なおこの際、FからCに行くルートもあるが、Hから遠のくポイントは候補に入れていない。



この中で最小のコストでいけるのはA→D→F→Hであり、最短経路はA–D–F–Hとなる。



このようにしてダイクストラを解き、この実装が上記のソースコードである。
今回はboostライブラリに入っているdijkstraを実行して実行速度の比較と行ってみたいと思う。
以下にboostを用いたdijkstraの実装を記す。

対応表

S	A	B	C	D	E	F
0	1	2	3	4	5	6

・ソースコード

```
#include <boost/assign/list_of.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
#include <deque>
#include <iostream>
#include <string>
#include <vector>

typedef boost::adjacency_list<boost::listS, boost::vecS,
boost::directedS,
                              boost::no_property,
boost::property<boost::edge_weight_t, int>>
    Graph;
typedef std::pair<int, int> Edge;
typedef boost::graph_traits<Graph>::vertex_descriptor Vertex;

enum
{
    S,
    A,
    B,
    C,
    D,
    E,
    F,
    N
};
const std::string Names = "SABCDEF";

// グラフを作る
Graph make_graph()
{
    const std::vector<Edge> edges =
boost::assign::list_of<Edge>(S, A)(S, C)(S, B)(A, C)(A, D)(B,
C)(B, E)(C, F)(D, F)(E, F);

    const std::vector<int> weights =
boost::assign::list_of(30)(10)(15)(25)(60)(40)(20)(35)(20)(30);
```

```
    return Graph(edges.begin(), edges.end(), weights.begin(), N);
}

int main()
{
    const Graph g = make_graph();
    const Vertex from = S; // 開始地点
    const Vertex to = F;   // 目的地

    std::vector<Vertex> parents(boost::num_vertices(g));
    std::vector<std::size_t> distance(boost::num_vertices(g));

    // 最短経路を計算
    boost::dijkstra_shortest_paths(g, from,

boost::predecessor_map(&parents[0]).distance_map(&distance[0]));

    // 経路なし
    if (parents[to] == to)
    {
        std::cout << "no path" << std::endl;
        return 1;
    }

    // 最短経路の頂点リストを作成
    std::deque<Vertex> route;
    for (Vertex v = to; v != from; v = parents[v])
    {
        route.push_front(v);
    }
    route.push_front(from);

    // 経路の長さを計算
    const std::size_t n = distance[to];
    std::cout << "route length:" << n << std::endl;

    // 最短経路を出力
    for (const Vertex v : route)
```

```
    {
        std::cout << Names[v] << std::endl;
    }
}
```

・実行結果

```
10kaoru12@dot1x7553 ~/B/G/2/R/consideration> cd "/Users/10kaoru12/Box Sync/GitHub/2y_university_programming_report/Report_Task_1/consideration/" && g++ boost.cpp -I/usr/local/include -L/usr/local/lib -o boost && "/Users/10kaoru12/Box Sync/GitHub/2y_university_programming_report/Report_Task_1/consideration/"boost -O3 -mtune=native -march=native -std=c++17
route length:45
S
C
F
```

・実行速度比較（自作のダイクストラ）

※今回の実行速度比較にはstandard libraryのchronoを使用する。

```
10kaoru12@dot1x7553 ~/B/G/2/R/consideration> cd "/Users/10kaoru12/Box Sync/GitHub/2y_university_programming_report/Report_Task_1/consideration/" && g++ consideration.cpp -I/usr/local/include -L/usr/local/lib -o consideration && "/Users/10kaoru12/Box Sync/GitHub/2y_university_programming_report/Report_Task_1/consideration/"consideration -O3 -mtune=native -march=native -std=c++17
7 10
0 1 30
0 3 10
0 2 15
1 3 25
1 4 60
2 3 40
2 5 20
3 6 35
4 6 20
5 6 30
0 6
distance:45
19856 milli sec
```

・実行速度比較（自作のダイクストラopenMPを使用して並列化）

※今回の実行速度比較にはstandard libraryのchronoを使用する。

```
10kaoru12@dot1x7553 ~/B/G/2/R/consideration> cd "/Users/10kaor
u12/Box Sync/GitHub/2y_university_programming_report/Report_Ta
sk_1/consideration/" && g++ consideration.cpp -I/usr/local/inc
lude -L/usr/local/lib -o consideration && "/Users/10kaoru12/Bo
x Sync/GitHub/2y_university_programming_report/Report_Task_1/c
onsideration/"consideration -O3 -mtune=native -march=native -s
td=c++17 -fopenmp
7 10
0 1 30
0 3 10
0 2 15
1 3 25
1 4 60
2 3 40
2 5 20
3 6 35
4 6 20
5 6 30
0 6
distance:45
1978 milli sec
```

・実行速度比較（boostのダイクストラ）

※今回の実行速度比較にはSTDのchronoを使用する。

```
10kaoru12@dot1x7553 ~/B/G/2/R/consideration>
cd "/Users/10kaoru12/Box Sync/GitHub/2y_university_programming
_report/Report_Task_1/consideration/" && g++ boost.cpp -I/usr/
local/include -L/usr/local/lib -o boost && "/Users/10kaoru12/B
ox Sync/GitHub/2y_university_programming_report/Report_Task_1/
consideration/"boost -O3 -mtune=native -march=native -std=c++1
7
route length:45
S
C
F
0 milli sec
```

・説明

授業中に受けたグラフ理論について興味を持ったので、ダイクストラ法についての理解を深めるためにできるだけ高度なコンテナを使用せず、C言語で実装をした。
また、実行速度がとても遅いのでopenMPを使用して並列化を図り、高速化をした。
また、C++のboostライブラリにboost::graphにdijkstraが実装されているので、それを実行した。

今回の考察では、自作のダイクストラ<openMP<<boostという結果になった。