HW [6], [2/20/2019] **MCS 253P**

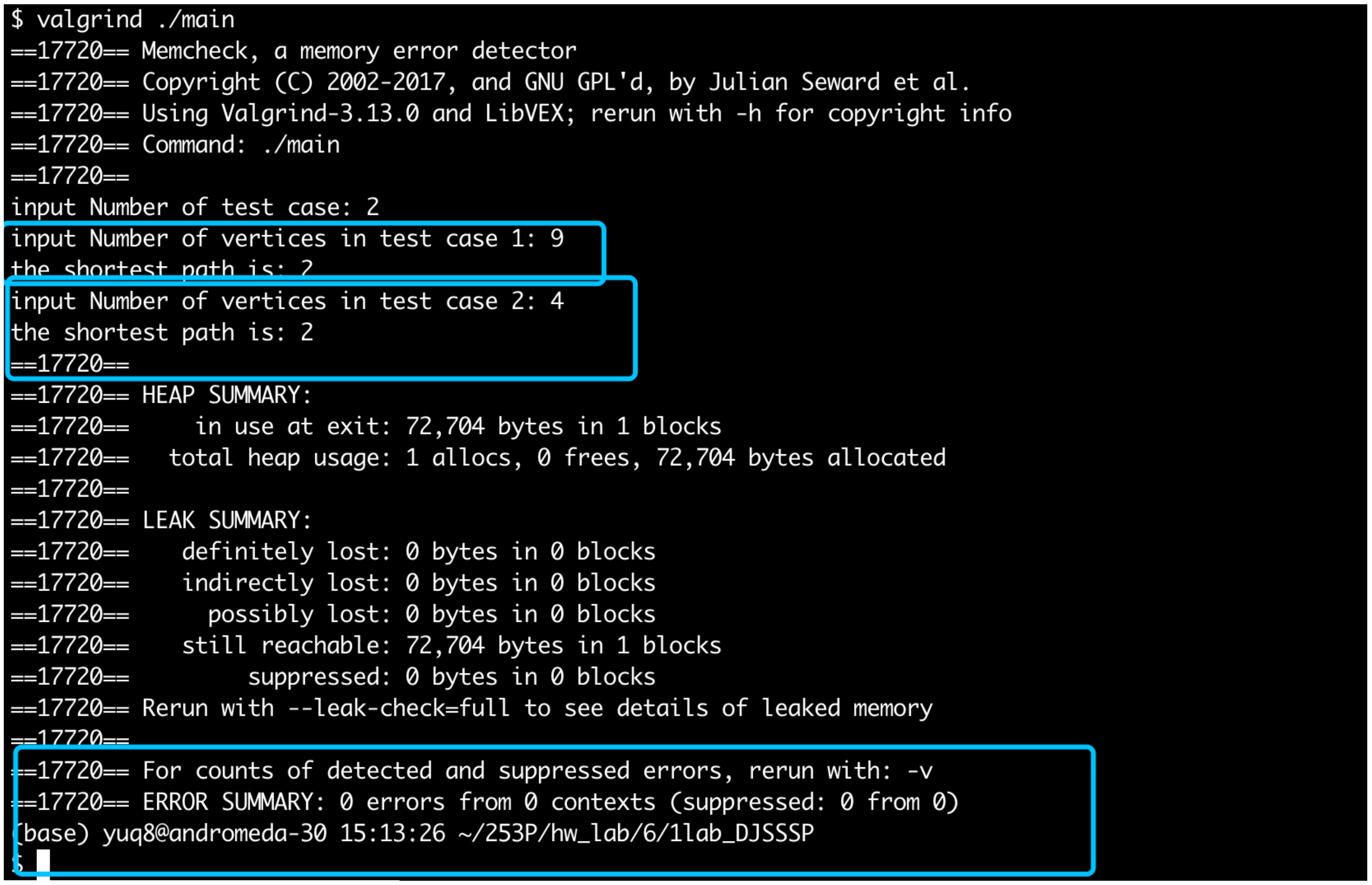
Name: Yu Qin

Results Screenshots

============================start of the write-up=====================

Lab 1: SSSP

test case 1,2,3:



```
$ valgrind ./main
==17720== Memcheck, a memory error detector
==17720== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==17720== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==17720== Command: ./main
==17720==
input Number of test case: 2
input Number of vertices in test case 1: 9
the shortest path is: 2
input Number of vertices in test case 2: 4
the shortest path is: 2
==17720==
==17720== HEAP SUMMARY:
==17720==     in use at exit: 72,704 bytes in 1 blocks
==17720==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==17720==
==17720== LEAK SUMMARY:
==17720==    definitely lost: 0 bytes in 0 blocks
==17720==    indirectly lost: 0 bytes in 0 blocks
==17720==      possibly lost: 0 bytes in 0 blocks
==17720==    still reachable: 72,704 bytes in 1 blocks
==17720==         suppressed: 0 bytes in 0 blocks
==17720== Rerun with --leak-check=full to see details of leaked memory
==17720==
==17720== For counts of detected and suppressed errors, rerun with: -v
==17720== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
(base) yuq8@andromeda-30 15:13:26 ~/253P/hw_lab/6/1lab_DJSSSP
$
```

Constraints:

1<=T<=30

1<=n <=1000

Example:

Input:

2

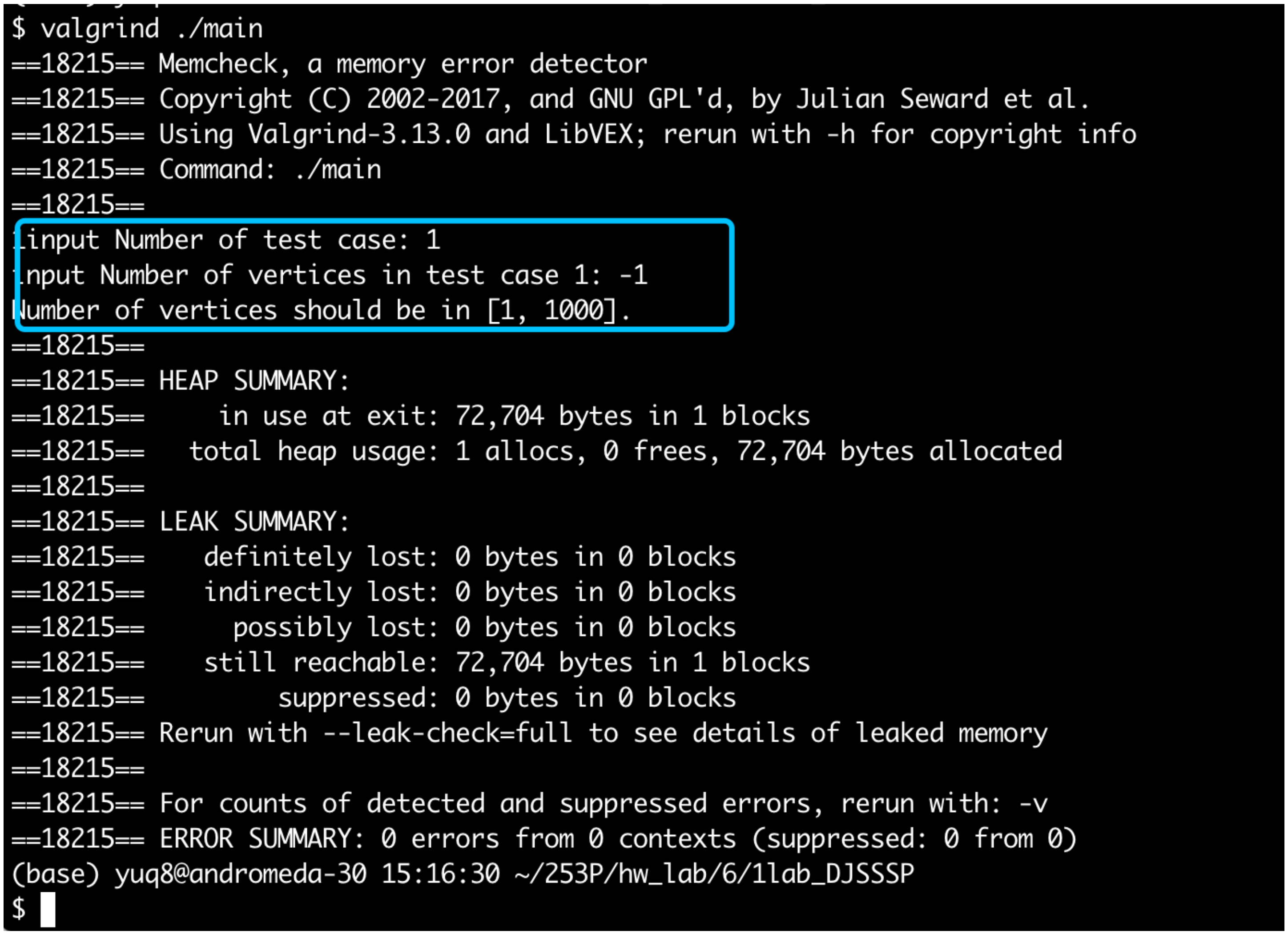
9

4

Output:

2

2

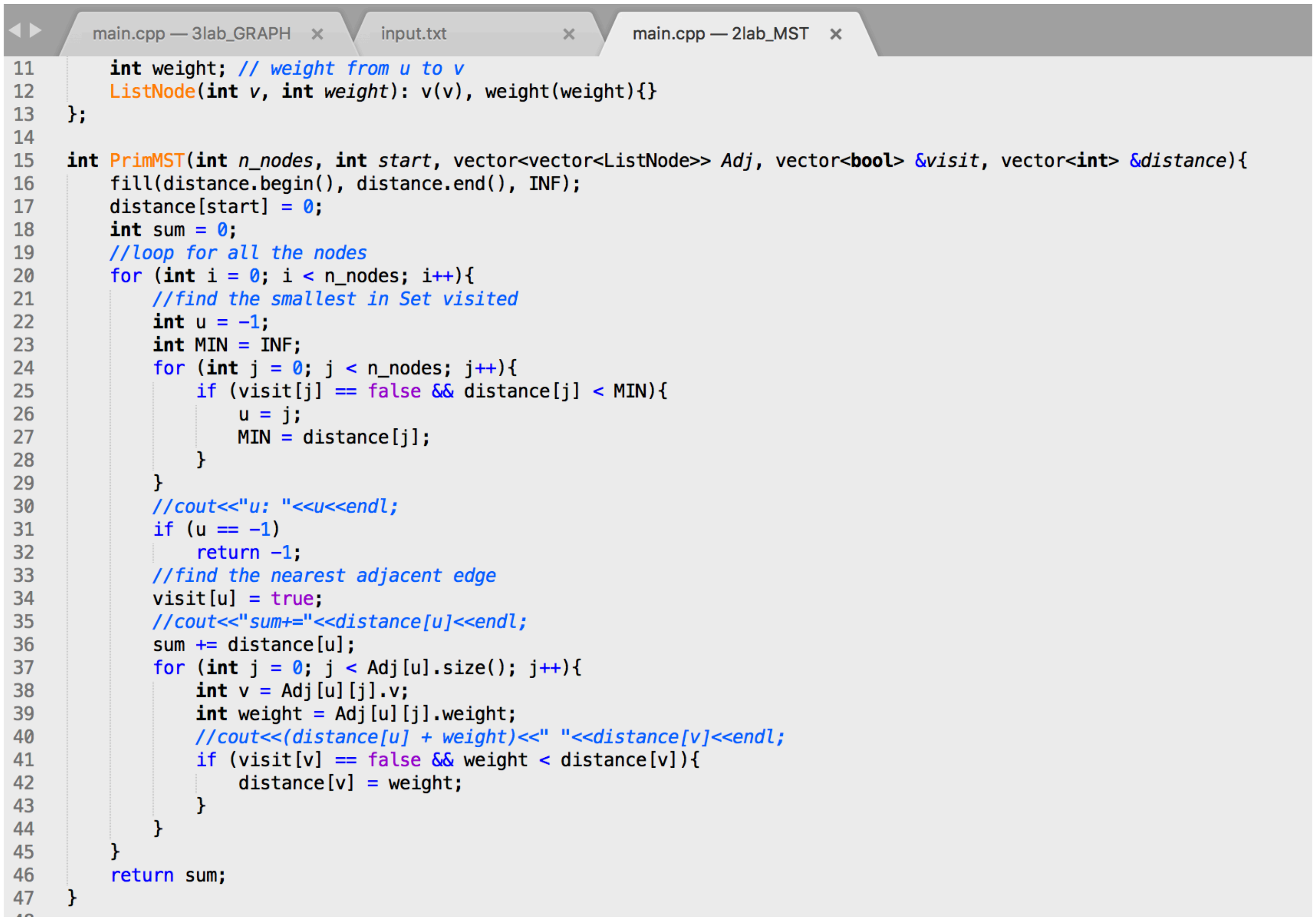


```
$ valgrind ./main
==18215== Memcheck, a memory error detector
==18215== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18215== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18215== Command: ./main
==18215==
input Number of test case: 1
input Number of vertices in test case 1: -1
Number of vertices should be in [1, 1000].
==18215==
==18215== HEAP SUMMARY:
==18215==     in use at exit: 72,704 bytes in 1 blocks
==18215==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==18215==
==18215== LEAK SUMMARY:
==18215==    definitely lost: 0 bytes in 0 blocks
==18215==    indirectly lost: 0 bytes in 0 blocks
==18215==      possibly lost: 0 bytes in 0 blocks
==18215==    still reachable: 72,704 bytes in 1 blocks
==18215==         suppressed: 0 bytes in 0 blocks
==18215== Rerun with --leak-check=full to see details of leaked memory
==18215==
==18215== For counts of detected and suppressed errors, rerun with: -v
==18215== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
(base) yuq8@andromeda-30 15:16:30 ~/253P/hw_lab/6/1lab_DJSSSP
$ 
```

Conclusion: right result, 0 error in memory leak.
Little Notes:
1) There are many ways to implement.

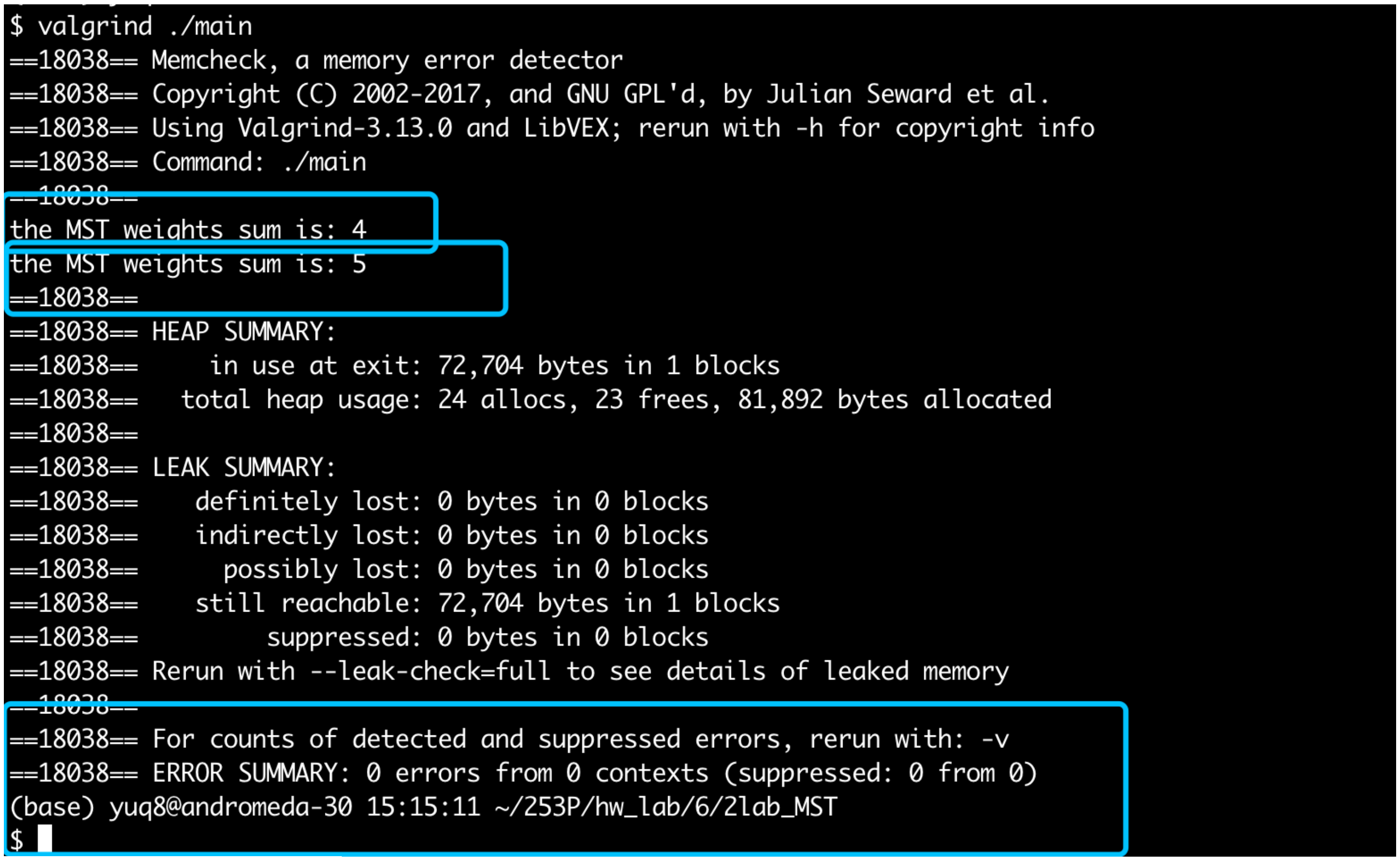
Lab 2: MST

Code:



```cpp
    int weight; // weight from u to v
    ListNode(int v, int weight): v(v), weight(weight){}
};

int PrimMST(int n_nodes, int start, vector<vector<ListNode>> Adj, vector<bool> &visit, vector<int> &distance){
    fill(distance.begin(), distance.end(), INF);
    distance[start] = 0;
    int sum = 0;
    //loop for all the nodes
    for (int i = 0; i < n_nodes; i++){
        //find the smallest in Set visited
        int u = -1;
        int MIN = INF;
        for (int j = 0; j < n_nodes; j++){
            if (visit[j] == false && distance[j] < MIN){
                u = j;
                MIN = distance[j];
            }
        }
        //cout<<"u: "<<u<<endl;
        if (u == -1)
            return -1;
        //find the nearest adjacent edge
        visit[u] = true;
        //cout<<"sum+="<<distance[u]<<endl;
        sum += distance[u];
        for (int j = 0; j < Adj[u].size(); j++){
            int v = Adj[u][j].v;
            int weight = Adj[u][j].weight;
            //cout<<(distance[u] + weight)<<" "<<distance[v]<<endl;
            if (visit[v] == false && weight < distance[v]){
                distance[v] = weight;
            }
        }
    }
    return sum;
}
```

Test cases 1, 2:



```
$ valgrind ./main
==18038== Memcheck, a memory error detector
==18038== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18038== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18038== Command: ./main
==18038==
the MST weights sum is: 4
the MST weights sum is: 5
==18038==
==18038== HEAP SUMMARY:
==18038==     in use at exit: 72,704 bytes in 1 blocks
==18038==   total heap usage: 24 allocs, 23 frees, 81,892 bytes allocated
==18038==
==18038== LEAK SUMMARY:
==18038==    definitely lost: 0 bytes in 0 blocks
==18038==    indirectly lost: 0 bytes in 0 blocks
==18038==      possibly lost: 0 bytes in 0 blocks
==18038==    still reachable: 72,704 bytes in 1 blocks
==18038==         suppressed: 0 bytes in 0 blocks
==18038== Rerun with --leak-check=full to see details of leaked memory
==18038==
==18038== For counts of detected and suppressed errors, rerun with: -v
==18038== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
(base) yuq8@andromeda-30 15:15:11 ~/253P/hw_lab/6/2lab_MST
$ 
```

Input

2

3 3

1 2 5 2 3 3 1 3 1

2 1

1 2 5

Output

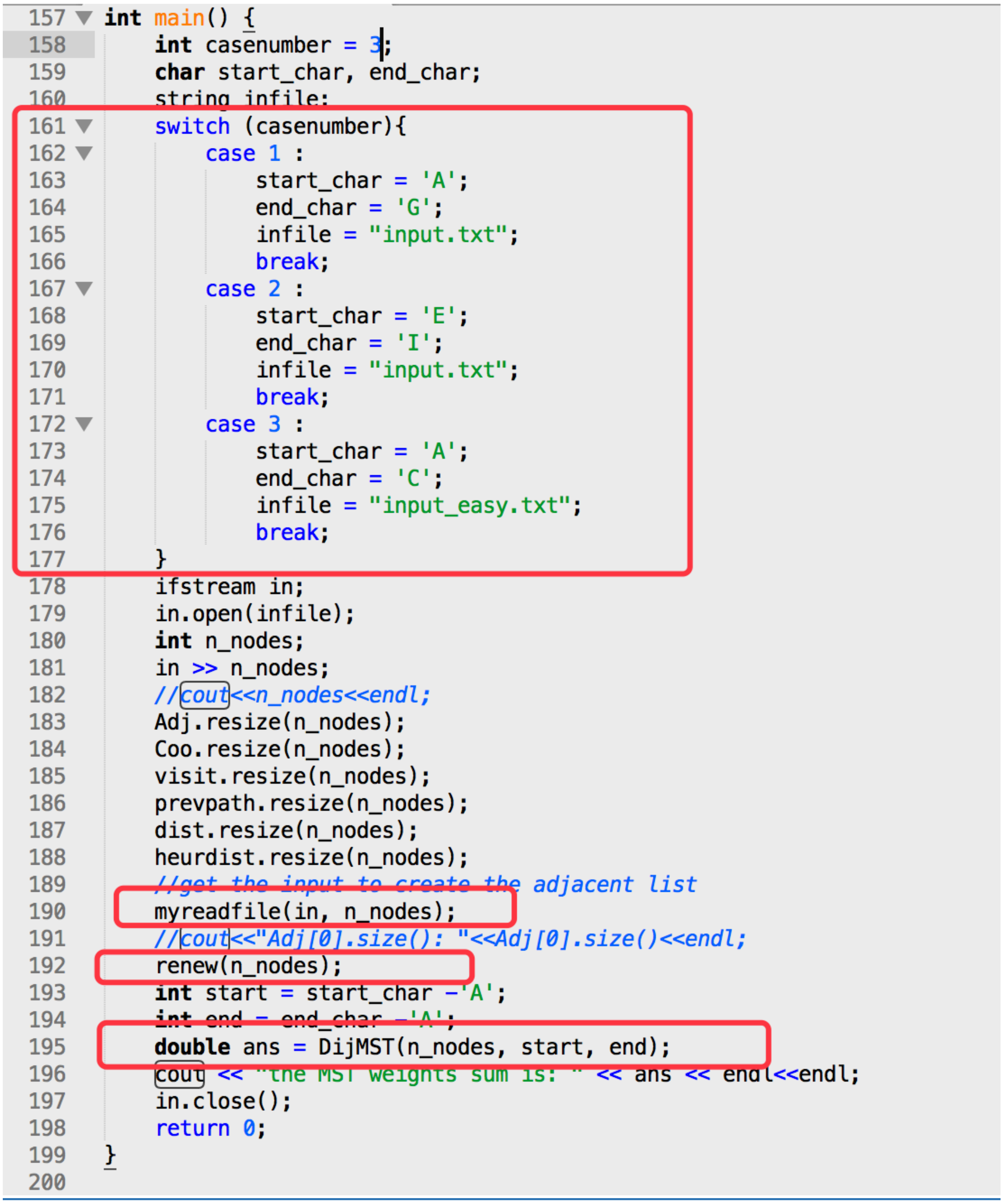
4

5

Conclusion: right result, 0 error in memory leak.
Little Notes:
1) Can be implemented in Prim and Krustal.

Lab3:

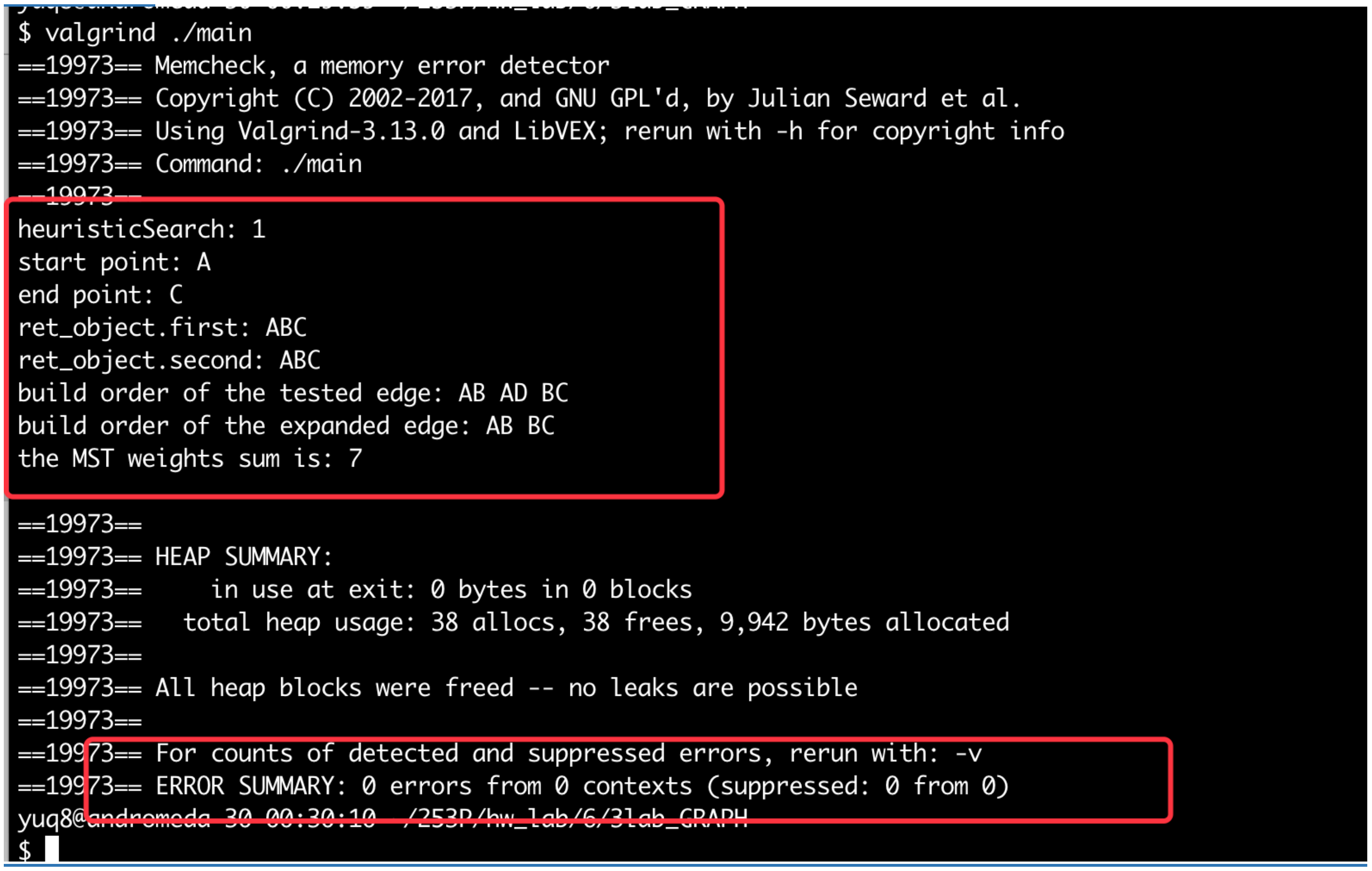
Code:



```
int main() {
    int casenumber = 3;
    char start_char, end_char;
    string infile;
    switch (casenumber){
        case 1 :
            start_char = 'A';
            end_char = 'G';
            infile = "input.txt";
            break;
        case 2 :
            start_char = 'E';
            end_char = 'I';
            infile = "input.txt";
            break;
        case 3 :
            start_char = 'A';
            end_char = 'C';
            infile = "input_easy.txt";
            break;
    }
    ifstream in;
    in.open(infile);
    int n_nodes;
    in >> n_nodes;
    //cout<<n_nodes<<endl;
    Adj.resize(n_nodes);
    Coo.resize(n_nodes);
    visit.resize(n_nodes);
    prevpath.resize(n_nodes);
    dist.resize(n_nodes);
    heurdist.resize(n_nodes);
    //get the input to create the adjacent list
    myreadfile(in, n_nodes);
    //cout<<"Adj[0].size(): "<<Adj[0].size()<<endl;
    renew(n_nodes);
    int start = start_char -'A';
    int end = end_char -'A';
    double ans = DijMST(n_nodes, start, end);
    cout << "the MST weights sum is: " << ans << endl<<endl;
    in.close();
    return 0;
}
```

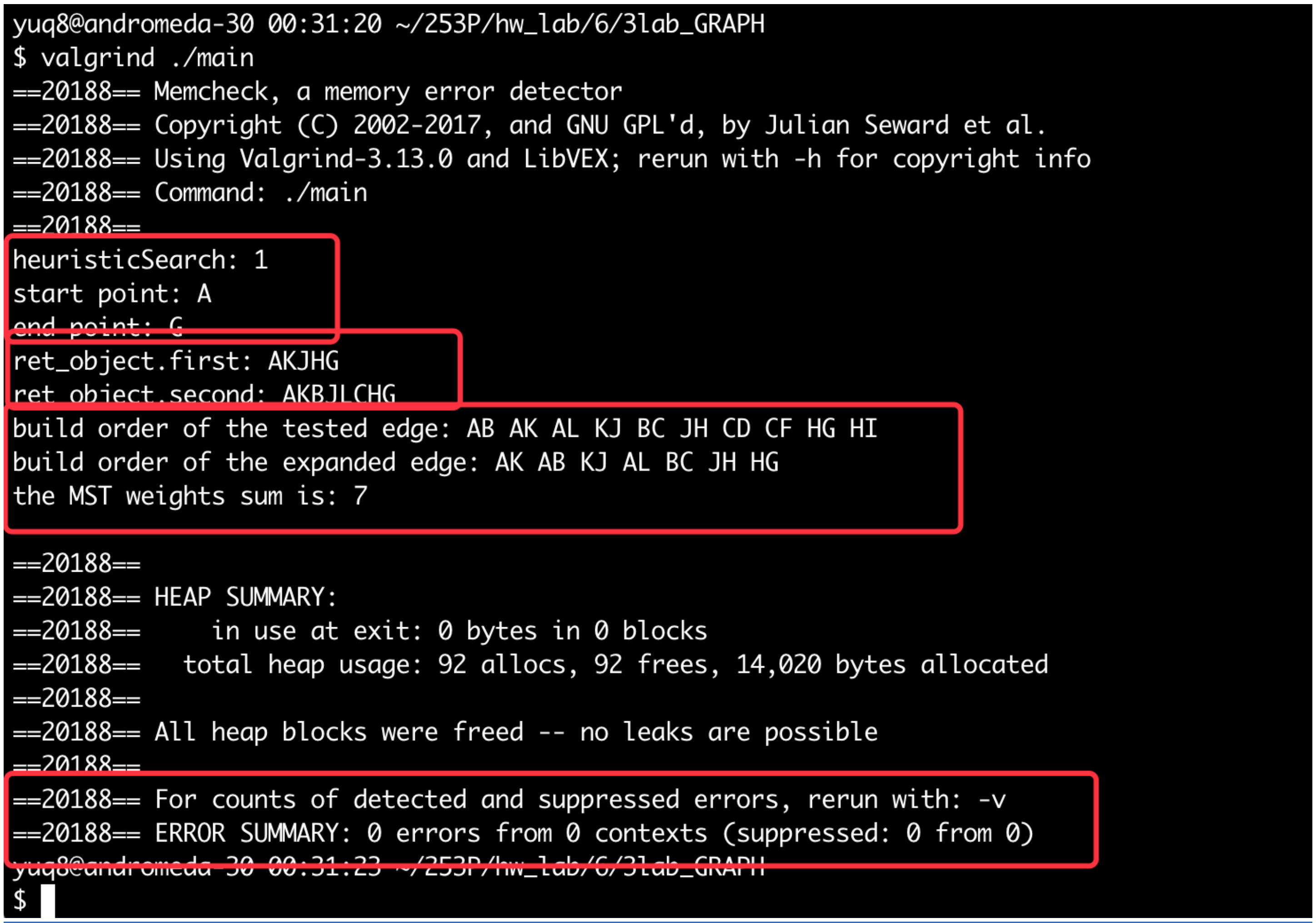
test cases 1: with easy input

```
4

A 4 3
B D

B 4 0
A C

C 0 0
B D

D 0 3
C A
```



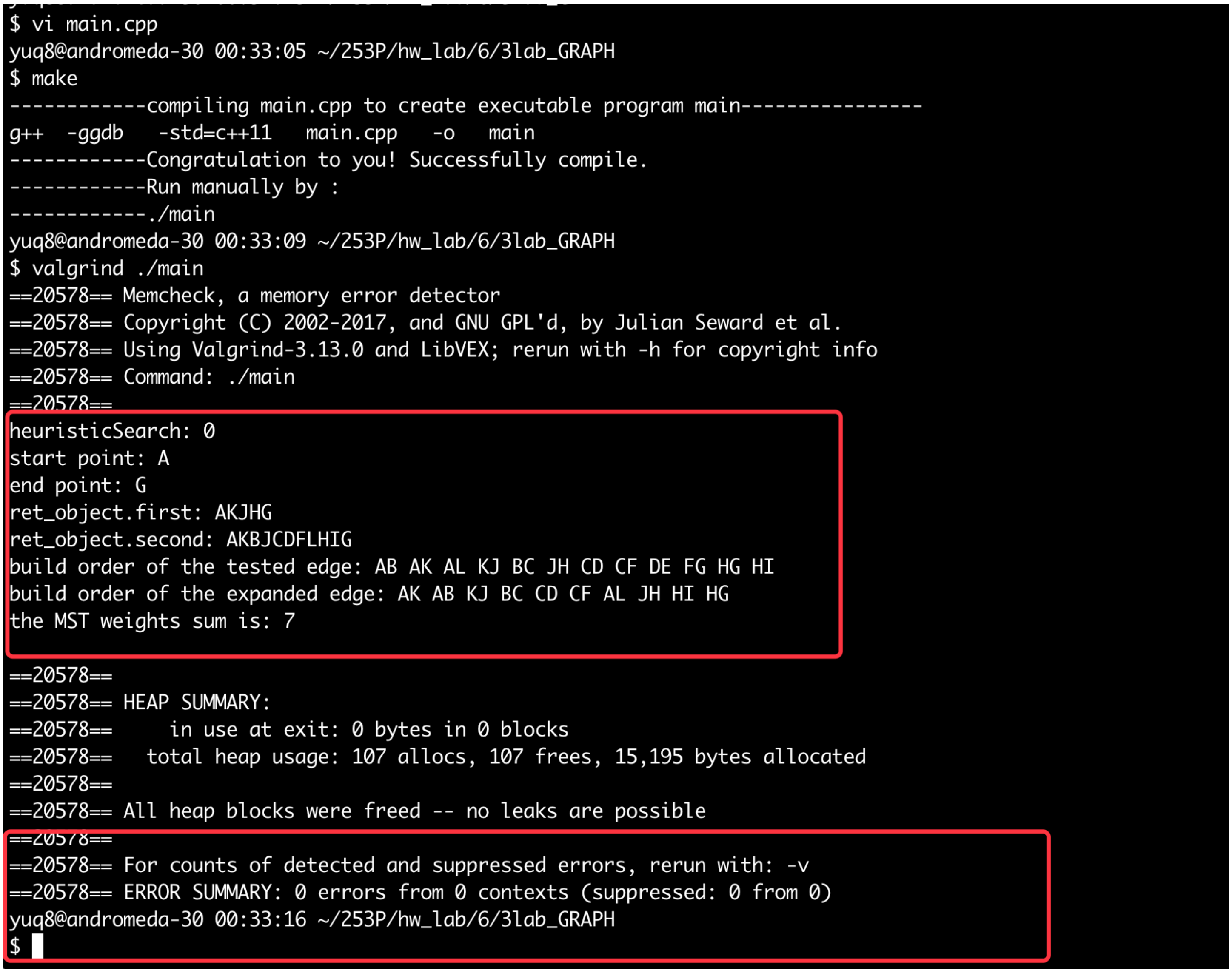
```
$ valgrind ./main
==19973== Memcheck, a memory error detector
==19973== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==19973== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==19973== Command: ./main
==19973==
heuristicSearch: 1
start point: A
end point: C
ret_object.first: ABC
ret_object.second: ABC
build order of the tested edge: AB AD BC
build order of the expanded edge: AB BC
the MST weights sum is: 7

==19973==
==19973== HEAP SUMMARY:
==19973==     in use at exit: 0 bytes in 0 blocks
==19973==   total heap usage: 38 allocs, 38 frees, 9,942 bytes allocated
==19973==
==19973== All heap blocks were freed -- no leaks are possible
==19973==
==19973== For counts of detected and suppressed errors, rerun with: -v
==19973== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda 30 00:30:10 ~/253P/hw_lab/6/3lab_GRAPH
$ 
```

test cases 2 with standard given input:
with heuri:



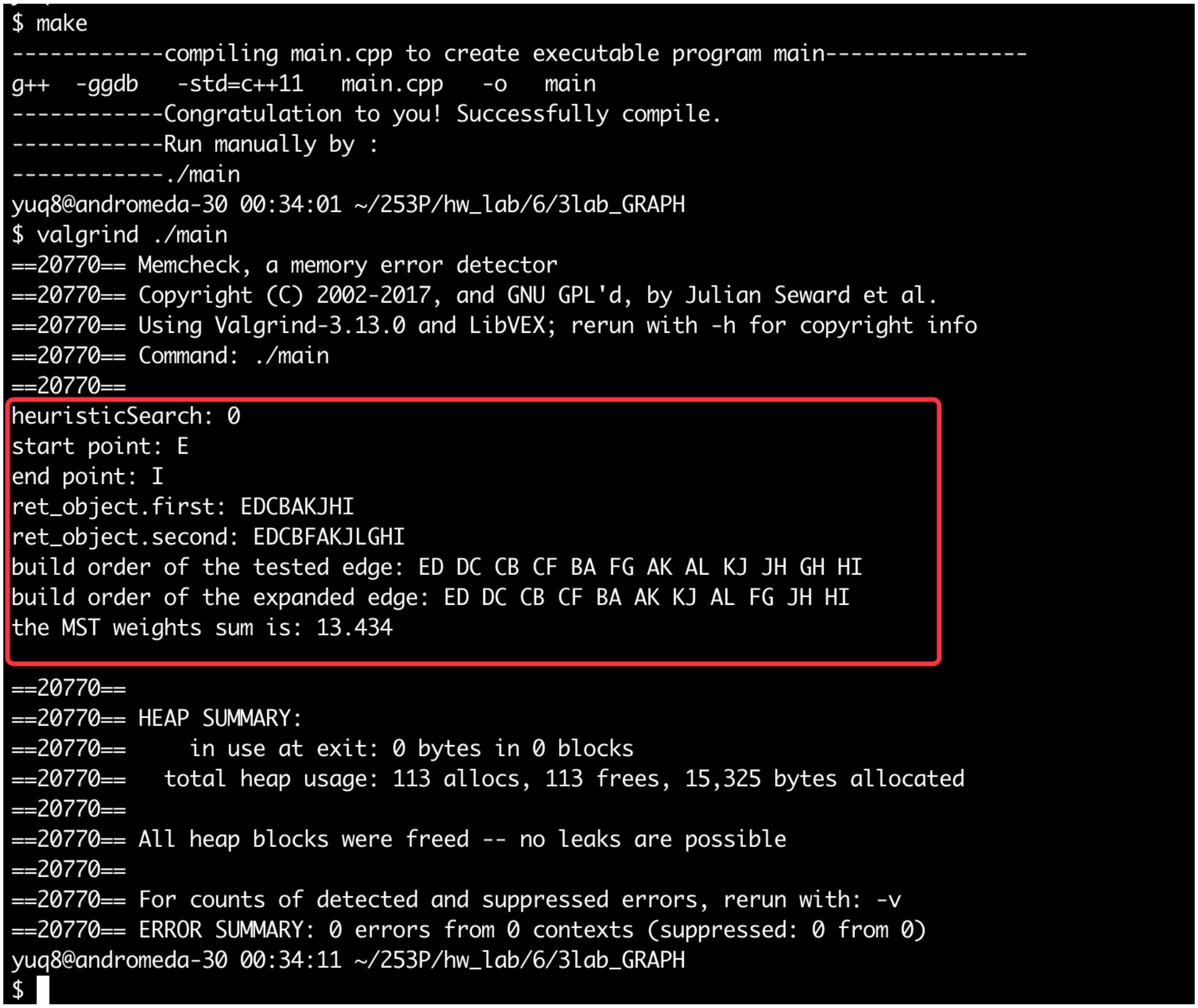
```
yuq8@andromeda-30 00:31:20 ~/253P/hw_lab/6/3lab_GRAPH
$ valgrind ./main
==20188== Memcheck, a memory error detector
==20188== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20188== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20188== Command: ./main
==20188==
heuristicSearch: 1
start point: A
end point: G
ret_object.first: AKJHG
ret_object.second: AKBJLCHG
build order of the tested edge: AB AK AL KJ BC JH CD CF HG HI
build order of the expanded edge: AK AB KJ AL BC JH HG
the MST weights sum is: 7

==20188==
==20188== HEAP SUMMARY:
==20188==     in use at exit: 0 bytes in 0 blocks
==20188==   total heap usage: 92 allocs, 92 frees, 14,020 bytes allocated
==20188==
==20188== All heap blocks were freed -- no leaks are possible
==20188==
==20188== For counts of detected and suppressed errors, rerun with: -v
==20188== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda-30 00:31:23 ~/253P/hw_lab/6/3lab_GRAPH
$
```

without heuri:



```
$ vi main.cpp
yuq8@andromeda-30 00:33:05 ~/253P/hw_lab/6/3lab_GRAPH
$ make
------------compiling main.cpp to create executable program main----------------
g++  -ggdb   -std=c++11   main.cpp   -o   main
------------Congratulation to you! Successfully compile.
------------Run manually by :
------------./main
yuq8@andromeda-30 00:33:09 ~/253P/hw_lab/6/3lab_GRAPH
$ valgrind ./main
==20578== Memcheck, a memory error detector
==20578== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20578== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20578== Command: ./main
==20578==
heuristicSearch: 0
start point: A
end point: G
ret_object.first: AKJHG
ret_object.second: AKBJCDFLHIG
build order of the tested edge: AB AK AL KJ BC JH CD CF DE FG HG HI
build order of the expanded edge: AK AB KJ BC CD CF AL JH HI HG
the MST weights sum is: 7

==20578==
==20578== HEAP SUMMARY:
==20578==     in use at exit: 0 bytes in 0 blocks
==20578==   total heap usage: 107 allocs, 107 frees, 15,195 bytes allocated
==20578==
==20578== All heap blocks were freed -- no leaks are possible
==20578==
==20578== For counts of detected and suppressed errors, rerun with: -v
==20578== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda-30 00:33:16 ~/253P/hw_lab/6/3lab_GRAPH
$ 
```

test 4:

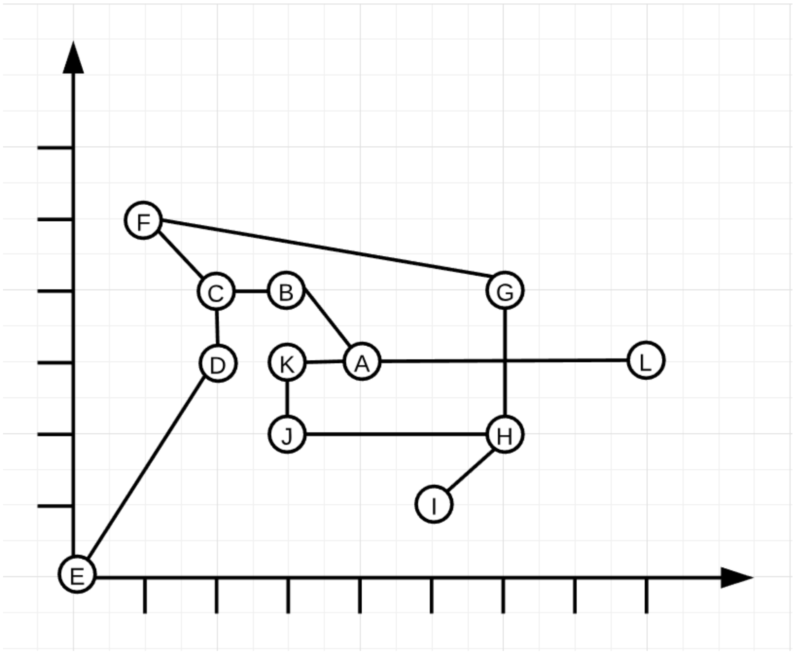


```
$ make
------------compiling main.cpp to create executable program main----------------
g++  -ggdb   -std=c++11   main.cpp   -o   main
------------Congratulation to you! Successfully compile.
------------Run manually by :
------------./main
yuq8@andromeda-30 00:34:01 ~/253P/hw_lab/6/3lab_GRAPH
$ valgrind ./main
==20770== Memcheck, a memory error detector
==20770== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20770== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20770== Command: ./main
==20770==
heuristicSearch: 0
start point: E
end point: I
ret_object.first: EDCBAKJHI
ret_object.second: EDCBFAKJLGHI
build order of the tested edge: ED DC CB CF BA FG AK AL KJ JH GH HI
build order of the expanded edge: ED DC CB CF BA AK KJ AL FG JH HI
the MST weights sum is: 13.434

==20770==
==20770== HEAP SUMMARY:
==20770==     in use at exit: 0 bytes in 0 blocks
==20770==   total heap usage: 113 allocs, 113 frees, 15,325 bytes allocated
==20770==
==20770== All heap blocks were freed -- no leaks are possible
==20770==
==20770== For counts of detected and suppressed errors, rerun with: -v
==20770== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda-30 00:34:11 ~/253P/hw_lab/6/3lab_GRAPH
$ 
```

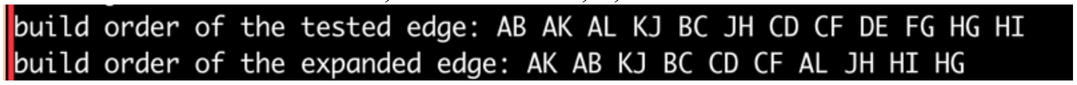
Part II (50pts) - [REQUIRES COMPLETION OF PART I FOR CREDIT]

Copy the image of the graph for the standard input twice.

- for one, label the edges in order (1, 2, 3, ...) as they were established during Dijkstra's
- for the other, label the edges in order (1, 2, 3, ...) as they were established during A*

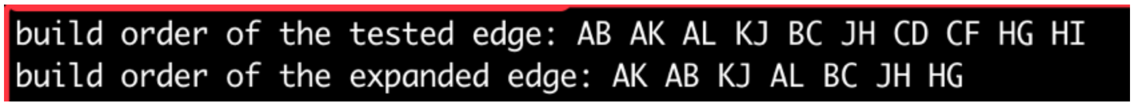


As shown in the code, the order are following. But since I'm not sure whether you mean to, I give both the refresh edge and expanded node:
For without heur from A to G, the order of 1, 2, 3… is :



```
build order of the tested edge: AB AK AL KJ BC JH CD CF DE FG HG HI
build order of the expanded edge: AK AB KJ BC CD CF AL JH HI HG
```

For with heur from A to G:



```
build order of the tested edge: AB AK AL KJ BC JH CD CF HG HI
build order of the expanded edge: AK AB KJ AL BC JH HG
```

Since it is the same order as above, so I let alone the just drawing step.

Lab 5.3: All about Coins——special column

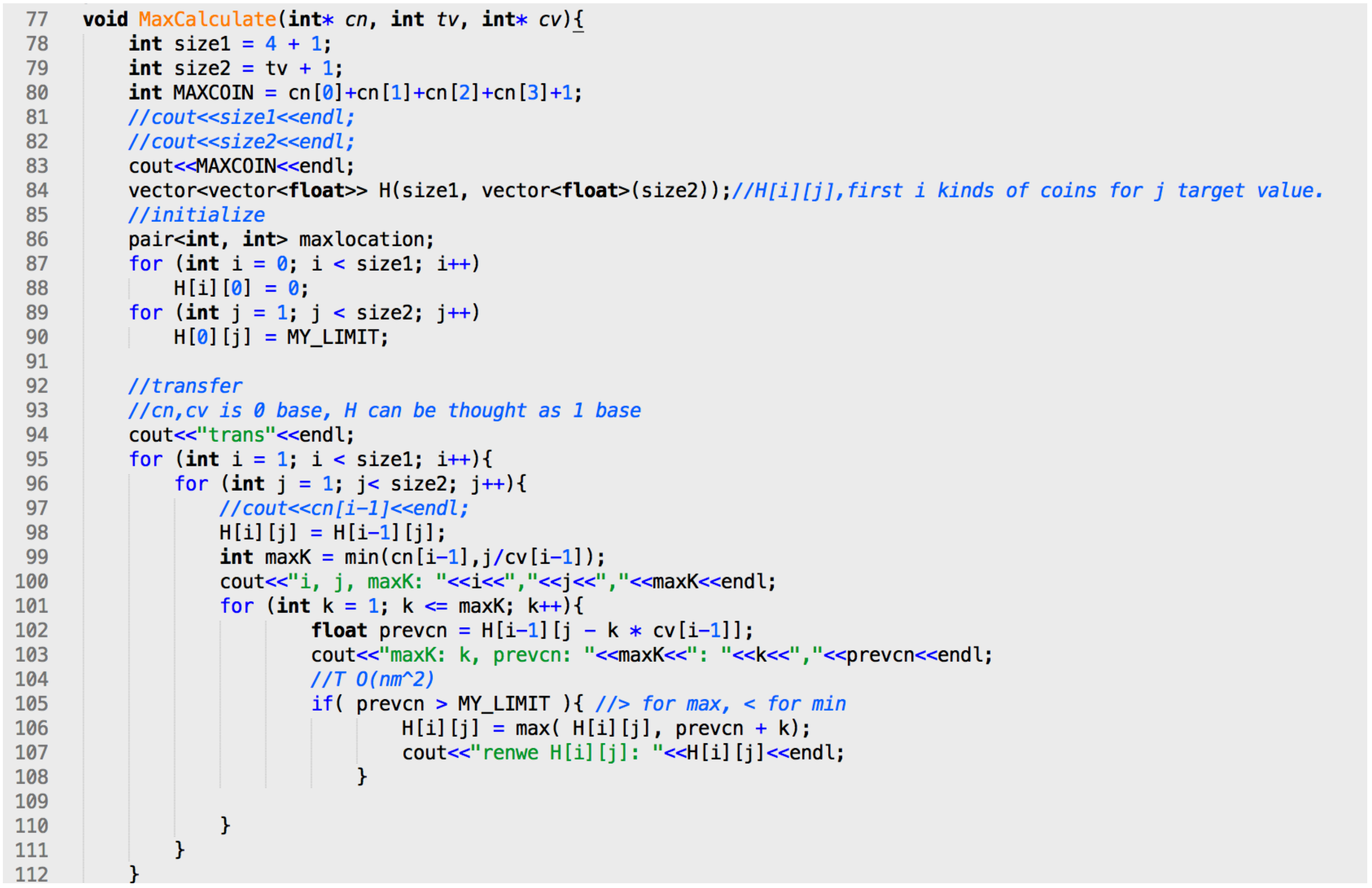
General Problem Description

About coins, there are many kinds of detailed categories. For example:

	Unlimited coins number	Limited coins number
Max coins		Lab5.3
Min coins	[1], LC 322[2]	[1]
Solution nums		
Require sth.		

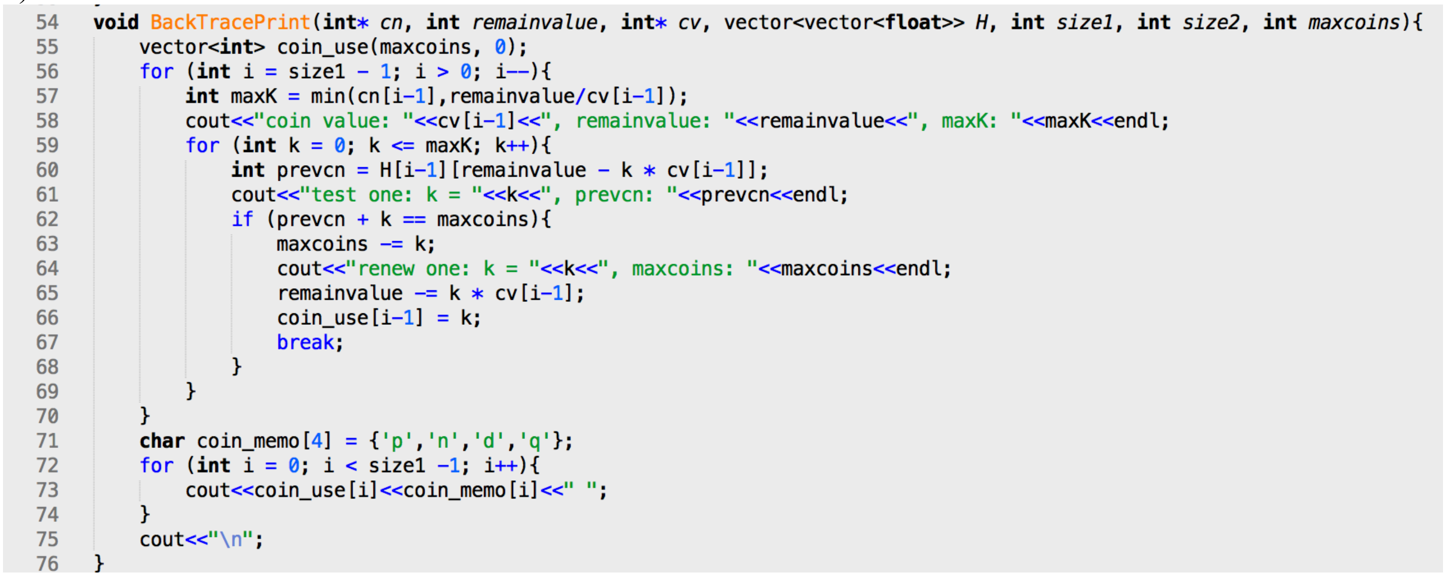
[1] https://blog.csdn.net/suwei19870312/article/details/9296415
[2] https://www.laioffer.com/zh/videos/2018-04-23-322-coin-change/

So, In this lab, I write code mainly consisting of 2 parts：
1) find the max in DP



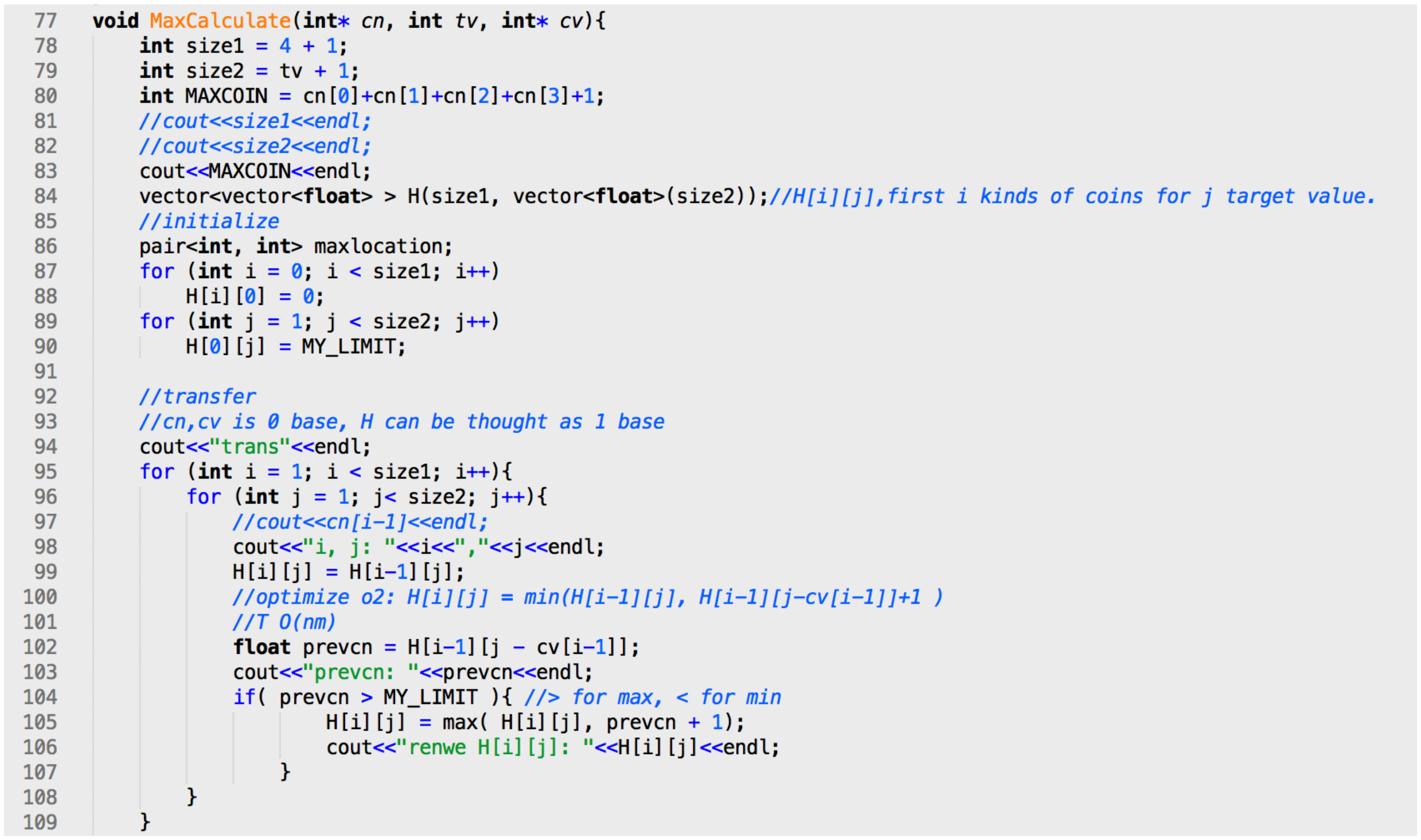
```cpp
void MaxCalculate(int* cn, int tv, int* cv){
    int size1 = 4 + 1;
    int size2 = tv + 1;
    int MAXCOIN = cn[0]+cn[1]+cn[2]+cn[3]+1;
    //cout<<size1<<endl;
    //cout<<size2<<endl;
    cout<<MAXCOIN<<endl;
    vector<vector<float>> H(size1, vector<float>(size2));//H[i][j],first i kinds of coins for j target value.
    //initialize
    pair<int, int> maxlocation;
    for (int i = 0; i < size1; i++)
        H[i][0] = 0;
    for (int j = 1; j < size2; j++)
        H[0][j] = MY_LIMIT;

    //transfer
    //cn,cv is 0 base, H can be thought as 1 base
    cout<<"trans"<<endl;
    for (int i = 1; i < size1; i++){
        for (int j = 1; j< size2; j++){
            //cout<<cn[i-1]<<endl;
            H[i][j] = H[i-1][j];
            int maxK = min(cn[i-1],j/cv[i-1]);
            cout<<"i, j, maxK: "<<i<<","<<j<<","<<maxK<<endl;
            for (int k = 1; k <= maxK; k++){
                    float prevcn = H[i-1][j - k * cv[i-1]];
                    cout<<"maxK: k, prevcn: "<<maxK<<": "<<k<<","<<prevcn<<endl;
                    //T O(nm^2)
                    if( prevcn > MY_LIMIT ){ //> for max, < for min
                        H[i][j] = max( H[i][j], prevcn + k);
                        cout<<"renwe H[i][j]: "<<H[i][j]<<endl;
                    }

            }
        }
    }
```

2) traceback:



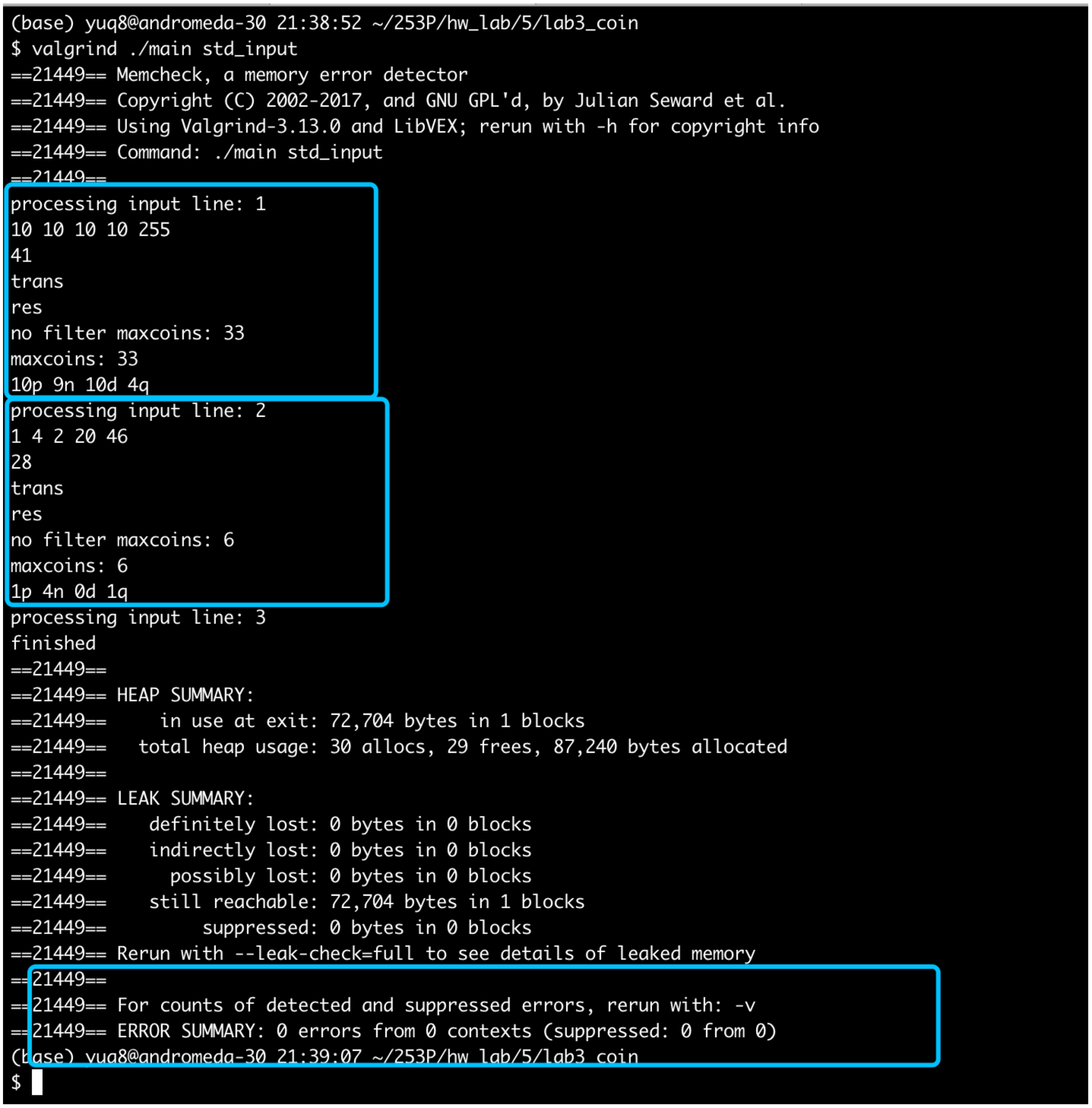
```cpp
void BackTracePrint(int* cn, int remainvalue, int* cv, vector<vector<float>> H, int size1, int size2, int maxcoins){
    vector<int> coin_use(maxcoins, 0);
    for (int i = size1 - 1; i > 0; i--){
        int maxK = min(cn[i-1],remainvalue/cv[i-1]);
        cout<<"coin value: "<<cv[i-1]<<", remainvalue: "<<remainvalue<<", maxK: "<<maxK<<endl;
        for (int k = 0; k <= maxK; k++){
            int prevcn = H[i-1][remainvalue - k * cv[i-1]];
            cout<<"test one: k = "<<k<<", prevcn: "<<prevcn<<endl;
            if (prevcn + k == maxcoins){
                maxcoins -= k;
                cout<<"renew one: k = "<<k<<", maxcoins: "<<maxcoins<<endl;
                remainvalue -= k * cv[i-1];
                coin_use[i-1] = k;
                break;
            }
        }
    }
    char coin_memo[4] = {'p','n','d','q'};
    for (int i = 0; i < size1 -1; i++){
        cout<<coin_use[i]<<coin_memo[i]<<" ";
    }
    cout<<"\n";
}
```

and the first part can be optimized in Time to :



```cpp
void MaxCalculate(int* cn, int tv, int* cv){
    int size1 = 4 + 1;
    int size2 = tv + 1;
    int MAXCOIN = cn[0]+cn[1]+cn[2]+cn[3]+1;
    //cout<<size1<<endl;
    //cout<<size2<<endl;
    cout<<MAXCOIN<<endl;
    vector<vector<float> > H(size1, vector<float>(size2));//H[i][j],first i kinds of coins for j target value.
    //initialize
    pair<int, int> maxlocation;
    for (int i = 0; i < size1; i++)
        H[i][0] = 0;
    for (int j = 1; j < size2; j++)
        H[0][j] = MY_LIMIT;

    //transfer
    //cn,cv is 0 base, H can be thought as 1 base
    cout<<"trans"<<endl;
    for (int i = 1; i < size1; i++){
        for (int j = 1; j< size2; j++){
            //cout<<cn[i-1]<<endl;
            cout<<"i, j: "<<i<<","<<j<<endl;
            H[i][j] = H[i-1][j];
            //optimize o2: H[i][j] = min(H[i-1][j], H[i-1][j-cv[i-1]]+1 )
            //T O(nm)
            float prevcn = H[i-1][j - cv[i-1]];
            cout<<"prevcn: "<<prevcn<<endl;
            if( prevcn > MY_LIMIT ){ //> for max, < for min
                H[i][j] = max( H[i][j], prevcn + 1);
                cout<<"renwe H[i][j]: "<<H[i][j]<<endl;
            }
        }
    }
```

test case 1, 2:



```
(base) yuq8@andromeda-30 21:38:52 ~/253P/hw_lab/5/lab3_coin
$ valgrind ./main std_input
==21449== Memcheck, a memory error detector
==21449== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==21449== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==21449== Command: ./main std_input
==21449==
processing input line: 1
10 10 10 10 255
41
trans
res
no filter maxcoins: 33
maxcoins: 33
10p 9n 10d 4q
processing input line: 2
1 4 2 20 46
28
trans
res
no filter maxcoins: 6
maxcoins: 6
1p 4n 0d 1q
processing input line: 3
finished
==21449==
==21449== HEAP SUMMARY:
==21449==     in use at exit: 72,704 bytes in 1 blocks
==21449==   total heap usage: 30 allocs, 29 frees, 87,240 bytes allocated
==21449==
==21449== LEAK SUMMARY:
==21449==    definitely lost: 0 bytes in 0 blocks
==21449==    indirectly lost: 0 bytes in 0 blocks
==21449==      possibly lost: 0 bytes in 0 blocks
==21449==    still reachable: 72,704 bytes in 1 blocks
==21449==         suppressed: 0 bytes in 0 blocks
==21449== Rerun with --leak-check=full to see details of leaked memory
==21449==
==21449== For counts of detected and suppressed errors, rerun with: -v
==21449== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
(base) yuq8@andromeda-30 21:39:07 ~/253P/hw_lab/5/lab3_coin
$ 
```

What's more, it can be optimized in space to O(m).

The result screenshot is:

And future similar codes are on going… to be continued.