Stony Brook University

Combinatorial Search

Homework #4

https://github.com/Fanngdai/Combinatorial-Search

If this repo is set to public, you may access it with the link above. It will not be set to public until after November 15th, 2018.

Fanng Dai - StudID

CSE373 Analysis of Algorithms

Prof. Steven Skiena

Thursday November 8th, 2018

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <limits.h>

#include <string.h>

int \*\*edges;

int amt\_vertex;

int amt\_edge;

int \*result;

int bandwidth;

int min;

/\*

\* Checks for the max bandwidth in this permutation

\*

\* Loops through the edges and checks in the permutation to find the

\* distance the difference of index of the 2 vertices. What you do

\* first is find the first value of the two and keep it as an index.

\* Then, you look for the second value of the edge and take the

\* difference. If the difference is greater than the bandwidth, we

\* would return immediately! We only want to find a smaller bandwidth.

\* We will continue until we have a bandwidth of less than or equal

\* to what we originally had.

\*/

int checkMax(int a[], int k){

int max = 0;

for(int e = 0; e < amt\_edge; e++) {

int index = -1;

int one = edges[e][0];

int two = edges[e][1];

for(int i=0; i<k; i++) {

if(index!=-1 && (a[i]==one || a[i]==two)) {

int tempMax = abs(index-i);

// Optimization

if(tempMax > bandwidth) {

return tempMax;

} else if(tempMax > max) {

max = tempMax;

}

break;

} else if(a[i]==one || a[i]==two) {

index = i;

}

}

}

return max;

}

/\*

\* Gets the max bandwidth of the current permutation and compare it

\* with the minimum bandwidth which we previous got. If the current

\* bandwidth is smaller what we have, we will replace the result

\* permutation along with a record of the new min bandwidth.

\*

\* You get the bandwidth of the current permutation and compare it to

\* the previous. We want the smaller bandwidth therefore, if the

\* current bandwidth is less than, we replace the bandwidth with this

\* permutation.

\*/

void process\_solution(int a[]) {

int max = checkMax(a, amt\_vertex);

if(max < bandwidth) {

memcpy(result, a, (amt\_vertex)\*sizeof(int));

bandwidth = max;

}

}

/\*

\* This function swaps the values of the two integer addresses.

\*/

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

/\*

\* Creates all the permutations. (Look at getMin() for the first

\* Optimization)

\* If we have a complete set, we will first check to see if this value

\* was already used.

\* A value (permutation) was already used if the last element is

\* greater than the first element. Notice 123 is the same as 321. If

\* we use 123, will be eliminated. Note how the first index is greater

\* than last. This will cut half the amount of time to

\* process\_solution.

\* If the set is not complete, I first checked to see if the

\* subset we have a bandwidth which is greater than the bandwidth

\* we already have. Since are searching for a smaller bandwidth, if

\* the subset has a greater bandwidth, we would stop this subset from

\* continuing and move to another subset.

\*/

void permute(int \*a,int i) {

// Optimization - look at getMin()

if(bandwidth==min)

return;

else if (amt\_vertex-1 == i){

// Optimization

if(a[0]<a[amt\_vertex-1]) {

process\_solution(a);

}

return;

}

// Optimization

if(checkMax(a, i)>bandwidth)

return;

for (int j = i; j<amt\_vertex; j++) {

swap(a+i,a+j);

permute(a,i+1);

swap(a+i,a+j);

}

}

/\*

\* I noticed that if a vertex is adjacent to x amount of vertex, then

\* the max bandwidth would be x/2. I searched for the vertex with the

\* most amount of degree. The value which I get (x/2) is the minimum

\* value the bandwidth can.

\*/

void getMin() {

int vals[amt\_vertex];

for(int i=0; i<amt\_vertex; i++)

vals[i]=0;

for(int i=0; i<amt\_edge; i++) {

int one = edges[i][0];

int two = edges[i][1];

vals[one-1]++;

vals[two-1]++;

}

min = 0;

for(int i=0; i<amt\_vertex; i++) {

printf("%d ", vals[i]);

if(vals[i]>min)

min = vals[i];

}

min += 1;

min /= 2;

}

/\*

\* I do not calculate the time I spent reading the file and setting up

\* my program since there are many ways of doing it such as

\* hardcoding. Since everyone did it differently and some can be O(1)

\* time, I chose to leave it out. My edges are in a 2d array which

\* always has a column of 2 and a row of the amount of edges.

\*/

int main() {

char \*filename = "./Samples/g-t-8-7";

FILE \*file = fopen(filename, "r");

if(file == NULL)

exit(0);

fscanf(file, "%d\n%d", &amt\_vertex, &amt\_edge);

result = (int \*)malloc(amt\_vertex\* sizeof(int));

edges = (int \*\*)malloc(amt\_edge\*sizeof(int \*));

for(int i=0; i<amt\_edge; i++)

edges[i] = (int \*)malloc(2 \* sizeof(int));

bandwidth = INT\_MAX;

int temp\_amt\_edge = 0;

while(temp\_amt\_edge != amt\_edge) {

int temp\_from;

int temp\_to;

int temp = fscanf(file, "%d %d", &temp\_from, &temp\_to);

// If there are 2 numbers

if(temp == 2){

edges[temp\_amt\_edge][0] = temp\_from;

edges[temp\_amt\_edge][1] = temp\_to;

}

temp\_amt\_edge++;

}

int a[amt\_vertex];

for(int i=0; i<amt\_vertex; i++)

a[i] = i+1;

clock\_t start, end;

start = clock();

getMin();

permute(a, 0);

end = clock();

printf("Filename: %s\n", filename);

printf("Max size: %d\n", bandwidth);

for(int i=0; i<amt\_vertex; i++) {

printf("%d ", result[i]);

}

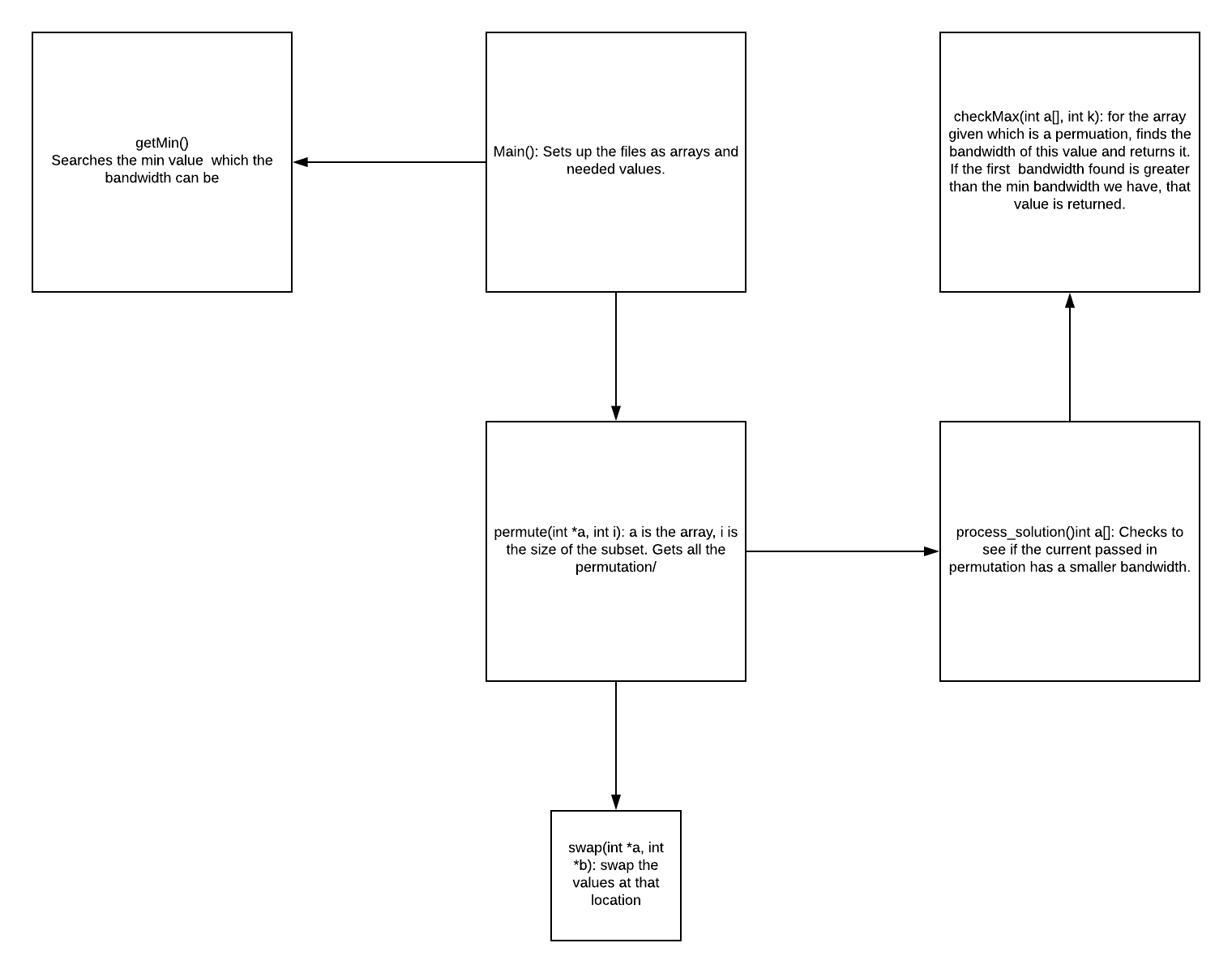
printf("\nTotal time taken by CPU: %f\n", (double)(end-start)/CLOCKS\_PER\_SEC);

fclose(file);

free(edges);

free(result);

}



**Optimizations I used**

1. When checking for the bandwidth of a permutation, if the bandwidth found is greater than the min bandwidth we have, it will end the function. By ending the function, it will stop looking for bandwidth. We know that this permutation does not have the minimum bandwidth we are searching for.
2. getMin() – This may not shorten the program, but if it does, it will shorten it by a lot. This function will figure out the minimum size of bandwidth can be. Our bandwidth result can be larger, but if is the min size (read description of getMin() function for this) it will return immediately with the permutation we want.
3. We remove the reverse permutation cutting the time complexity by at least half. 123 is the same as 321.
4. If a subset of the permutation has a bandwidth of greater than the min bandwidth we have(which we already calculated with another permutation) then we should not use this subset.

**Following are all files that run under 1 minute**

Filename: **g-bt-10-9**

Max size: 2

6 3 7 1 5 2 10 4 9 8

Total time taken by CPU: 0.103894

Filename: **g-bt-11-10**

Max size: 2

6 7 3 8 1 4 2 9 5 10 11

Total time taken by CPU: 0.959074

Filename: **g-bt-12-11**

Max size: 2

8 9 4 10 2 5 1 11 3 6 7 12

Total time taken by CPU: 7.915316

Filename: **g-bt-13-12**

Max size: 2

8 9 4 10 2 5 1 11 3 7 6 12 13

Total time taken by CPU: 45.928029

Filename: **g-gg-9-12**

Max size: 3

1 4 9 3 7 2 5 6 8

Total time taken by CPU: 0.040316

Filename: **g-gg-12-17**

Max size: 3

2 6 10 11 7 12 1 3 8 5 9 4

Total time taken by CPU: 5.374951

Filename: **g-p-10-9**

Max size: 1

1 5 6 4 7 2 10 8 9 3

Total time taken by CPU: 0.008809

Filename: **g-p-11-10**

Max size: 1

1 3 7 8 9 2 4 5 10 6 11

Total time taken by CPU: 0.030680

Filename: **g-p-12-11**

Max size: 1

9 8 7 2 6 4 3 5 1 10 11 12

Total time taken by CPU: 1.915173

Filename: **g-p-13-12**

Max size: 1

5 8 3 1 4 12 7 13 2 10 9 6 11

Total time taken by CPU: 4.531570

Filename: **g-p-14-13**

Max size: 1

7 8 10 3 2 5 6 14 11 12 9 1 4 13

Total time taken by CPU: 36.326090

Filename: **g-r-7-7**

Max size: 2

2 6 1 5 7 3 4

Total time taken by CPU: 0.000606

Filename: **g-r-8-4**

Max size: 1

1 7 3 6 5 4 2 8

Total time taken by CPU: 0.000460

Filename: **g-r-8-6**

Max size: 2

2 6 7 1 5 4 3 8

Total time taken by CPU: 0.001914

Filename: **g-r-8-11**

Max size: 3

4 3 5 7 2 1 6 8

Total time taken by CPU: 0.010907

Filename: **g-r-9-13**

Max size: 4

2 3 1 7 8 9 4 5 6

Total time taken by CPU: 0.119798

Filename: **g-r-10-9**

Max size: 2

2 3 6 7 4 1 5 9 10 8

Total time taken by CPU: 0.049072

Filename: **g-r-10-14**

Max size: 3

2 5 9 4 3 10 1 6 7 8

Total time taken by CPU: 0.239072

Filename: **g-r-11-15**

Max size: 4

1 3 8 9 4 2 6 5 11 7 10

Total time taken by CPU: 3.600711

Filename: **g-r-12-15**

Max size: 3

1 4 6 8 7 2 11 5 9 3 10 12

Total time taken by CPU: 5.878064

Filename: **g-r-13-15**

Max size: 3

1 6 9 7 13 11 5 10 3 4 12 2 8

Total time taken by CPU: 39.952443

Filename: **g-t-7-6**

Max size: 2

1 2 4 6 7 3 5

Total time taken by CPU: 0.000063

Filename: **g-t-8-7**

Max size: 2

2 4 8 1 5 6 7 3

Total time taken by CPU: 0.001809

Filename: **g-t-9-8**

Max size: 2

1 2 8 4 3 5 7 9 6

Total time taken by CPU: 0.025801

Filename: **g-t-10-9**

Max size: 2

1 6 2 5 10 7 8 9 3 4

Total time taken by CPU: 0.138894

Filename: **g-t-11-10**

Max size: 2

5 11 2 1 9 4 3 8 7 6 10

Total time taken by CPU: 1.158671

Filename: **g-t-12-11**

Max size: 2

1 3 9 7 8 11 5 2 4 12 6 10

Total time taken by CPU: 3.184647

Filename: **g-t-13-12**

Max size: 3

1 2 3 10 5 6 8 4 11 12 13 7 9

Total time taken by CPU: 74.085802