**ĐẠI HỌC BÁCH KHOA**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN**

**GIẢI THUẬT VÀ LẬP TRÌNH**

**Đề tài 17:**

**GIẢNG VIÊN HƯỚNG DẪN:**

**SINH VIÊN THỰC HIỆN: Trần Văn Trí**

**LỚP: 18T2 NHÓM:18N10A**

**Đà Nẵng, XX/2020**

MỤC LỤC

[MỤC LỤC i](#_Toc48908442)

[LỜI MỞ ĐẦU ii](#_Toc48908443)

[DANH MỤC HÌNH VẼ iii](#_Toc48908444)

[1. GIỚI THIỆU ĐỀ TÀI 1](#_Toc48908445)

[2. CƠ SỞ LÝ THUYẾT 1](#_Toc48908446)

[2.1. Ý tưởng 1](#_Toc48908447)

[2.2. Cơ sở lý thuyết 1](#_Toc48908448)

[3. TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN 1](#_Toc48908449)

[3.1. Phát biểu bài toán 1](#_Toc48908450)

[3.2. Cấu trúc dữ liệu 1](#_Toc48908451)

[3.3. Thuật toán 1](#_Toc48908452)

[4. CHƯƠNG TRÌNH VÀ KẾT QUẢ 1](#_Toc48908453)

[4.1. Tổ chức chương trình 1](#_Toc48908454)

[4.2. Ngôn ngữ cài đặt 1](#_Toc48908455)

[4.3. Kết quả 1](#_Toc48908456)

[4.3.1. Giao diện chính của chương trình 1](#_Toc48908457)

[4.3.2. Kết quả thực thi của chương trình 1](#_Toc48908458)

[4.3.3. Nhận xét 1](#_Toc48908459)

[5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 1](#_Toc48908460)

[5.1. Kết luận 1](#_Toc48908461)

[5.2. Hướng phát triển 2](#_Toc48908462)

[TÀI LIỆU THAM KHẢO 3](#_Toc48908463)

LỜI MỞ ĐẦU

Cấu trúc dữ liệu và Phân tích, thiết kế giải thuật là hai học phần rất quan trọng đối với người lập trình. Học phần này được xem như nền tảng của lập trình máy tính. Nó là cơ sở vững chắc để giải quyết nhiều bài toán, đồng thời cung cấp cho chúng ta hiểu biết về các giải thuật tác động lên dữ liệu, cũng như cách tổ chức dữ liệu hiệu quả để tối ưu bài toán.

Sau khi học xong học phần lí thuyết, em đã nghiên cứu và thực hiện đồ án này như là một cách để củng cố và mở rộng kiến thức. Thông qua quá trình thực hiện đồ án, chúng em đã nắm bắt được những kỹ thuật quan trọng của việc xây dựng cấu trúc dữ liệu và phân tích, thiết kế giải thuật sao cho tối ưu nhất.

Bài toán “**Trả tiền**” mà em nghiên cứu và trình bày trong báo cáo sau đây là một ví dụ. Em xin chân thành cảm ơn thầy Võ Đức Hoàng đã bổ sung ý tưởng và tận tình giúp đỡ chúng em thực hiện đồ án này. Em cũng rất mong nhận được sự góp ý từ phía thầy để bài làm của em được hoàn thiện hơn.

DANH MỤC HÌNH VẼ

Hình 1:Xuất ra màn hình …………………………….…Trang 2,20

Hình 2:Xuất ra file 2…………………………………Trang 3,20,21

Hình 3:Xuất ra file 3…………………………………Trang 3,20,22

Hình 4:Màn hình chính……………………………..……..Trang 21

# GIỚI THIỆU ĐỀ TÀI

Một quầy trả tiền có N loại tiền với các mệnh giá (giá trị tiền ghi trên tờ tiền) là A[1], A[2], ….A[N] ( Các A[i] là nguyên dương và khác nhau từng đôi). Giả thiết loại tiền mệnh giá A[i] có B[i] tờ (1 i N). Có M khách ( được đánh số hiệu từ 1 đến M ) cần lấy tiền. Số tiền khách j cần lấy là K[j], K[j] nguyên dương, 1 j M). Quy định rằng với mỗi khách hoặc quầy từ chối chưa trả tiền hoặc quầy phải trả đúng số tiền mà khách cần lấy.Dữ liệu vào được cho trong file văn bản có tên INP.TXT trong đó dòng đầu ghi giá trị N (N 10), dòng tiếp theo ghi các giá trị A[1], A[2], ….A[N], dòng tiếp theo ghi các giá trị B[1], B[2], ….B[N], sau đó là dòng ghi giá trị M (M 20), cuối cùng là dòng ghi các giá trị K[1], K[2], ….K[M], tất cả các giá trị đều nguyên dương. 1. Đọc file dữ liệu và đưa ra màn hình nội dung file dữ liệu ( theo thứ tự trên). 2. Tìm cách trả tiền sao cho trả được nhiều khách nhất. Thông báo kết quả ra file văn bản với tên OUT2.TXT trong đó dòng đầu ghi số khách được trả tiền, trong các dòng tiếp theo, mỗi dòng ghi thông tin về một khách được trả tiền gồm số hiệu của khách, số tiền phải trả và dãy các số X[1], X[2],…..X[N], trong đó X[i] là số tờ của loại tiền mệnh giá A[i], 1 i N, được trả cho khách. 3. Tìm cách trả tiền sao cho trả được nhiều tiền nhất. Thông báo kết quả ra file văn bản với tên OUT3.TXT trong đó dòng đầu ghi tổng số tiền đã trả được, trong các dòng tiếp theo, mỗi dòng ghi thông tin về một khách được trả tiền theo quy cách giống câu 2 Chú ý: Hai giá trị liền nhau trên một dòng của các file văn bản cách nhau ít nhất một dấu trắng.

# CƠ SỞ LÝ THUYẾT

## Ý tưởng

Như chúng ta đã biết, trong các quầy bán hàng họ luôn đau đầu với việc kinh doanh không biết để đạt được hiệu quả tối ưu nhất về mặt tài chính hoặc tối ưu nhất về việc đáp ứng số lượng khách hàng lớn nhất. Từ đó ý tưởng về bài toán “**Trả tiền**” ra đời để giải quyết bài toán đó.

## Cơ sở lý thuyết

-Sử dụng thuật toán dfs để số cách đổi n tiền thành số tiền cho mỗi người, sau đó tiếp tục dùng dfs để nối số cách của **n** người lại với 2 tùy chọn đối với mỗi người:

+Chọn

+Không chọn

-Từ đó đưa ra số lượng nhiều nhất khách hàng phù hợp với số lượng tiền bàn đầu.

# TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

## Phát biểu bài toán

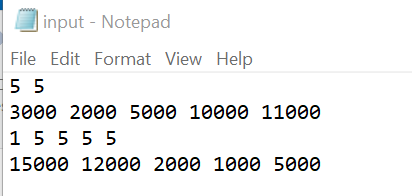
**INPUT** (Đầu vào):

N – số lượng loại tiền, M – số lượng khách hàng

A[i] (1<=i<=N) – giá trị tiền được ghi trên loại tiền thứ i

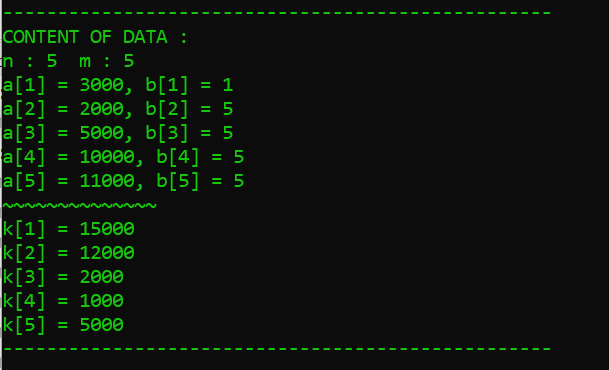
B[i] (1<=i<=N) – số lượng tiền của loại tiền thứ i

K[i] (1<=i<=M) – số tiền mà khách thứ i cần trả.



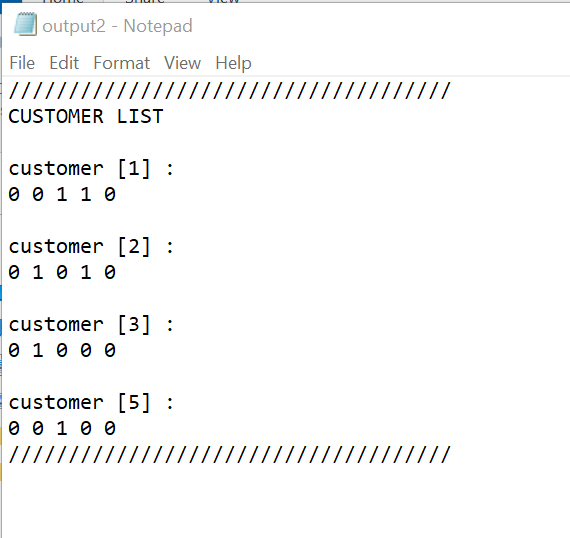
**OUTPUT** (Đầu ra):

-Đưa ra màn hình nội dung file dữ liệu

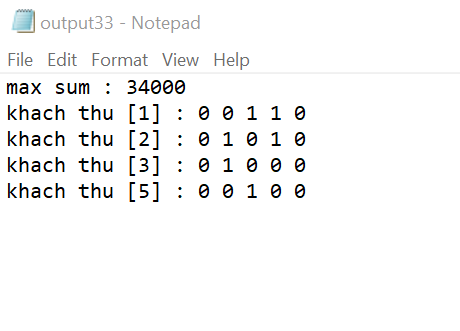


-In ra file số lượng khách được trả nhiều nhất.

+File 2 (In ra số lượng tiền sau khi trả cho khách)



+File 3 (In ra số lượng tiền phải trả cho từng khách)



## Cấu trúc dữ liệu

**vector**

|  |
| --- |
| template<typename T>  struct vector{  int n;  int capacity;  T \*a;  vector(){  this->n=0;  this->capacity=100;  a=new T[capacity];  }  vector(int size){  this->n=size;  this->capacity=100;  if(n>capacity) capacity=n;  a=new T[capacity];  }  void expand(int newCapacity){  if(newCapacity<=n){  return;  }  T \*old=a;  a=new T[newCapacity];  for(int i=0;i<n;++i) a[i]=old[i];  delete []old;  capacity=newCapacity;  }  int size() const{  return this->n;  }  T& operator[](const int id){  return a[id];  }  void resize(int size){  this->n=size;  capacity=100;  if(n>capacity) capacity=n;  this->a=new T[capacity];  }  void push\_back(T val){  if (n==capacity) expand(2\*n);  a[n]=val;  ++n;  }  void pop\_back(){  --n;  }  void clear(){  n=0;  }  }; |

**pair**

|  |
| --- |
| namespace PAIR{  template<typename T,typename U>  struct pair{  T first;  U second;  pair(){}  pair(T first,U second):first(first),second(second){}  };  } |

**customer**

|  |
| --- |
| **struct customer{**  **int id;**  **vector<int> x;**  **customer(){}**  **customer(int id,vector<int> x){**  **x.resize(n);**  **this->id=id;**  **this->x=x;**  **}**  **void print(ofstream& fo){**  **fo<<endl;**  **fo<<"customer ["<<this->id<<"] : "<<endl;**  **for(int i=0;i<x.size();++i){**  **fo<<x[i]<<" ";**  **}**  **fo<<endl;**  **}**  **void addVector(vector<int> a){**  **x.resize(a.size());**  **for(int i=0;i<a.size();++i) x[i]=a[i];**  **}**  **};** |

## Thuật toán

Với mỗi loại tiền thứ i t có B[i] số tiền tương đương với B[i] số cách chọn. Ta sẽ dung dfs để xâu chuỗi các trường hợp của mỗi loại tiền i sao cho i<=n. Sau đó nếu nó cho ra tổng phù hợp với người thứ j thì t sẽ bỏ trường hợp đó vào Tổng số trường hợp có thể có để trả được cho người j. Tương tự ta sẽ làm cho n người. Độ phức tạp của thuận này sẽ là O(n\*B[i]^n).

Sau khi thực ta tiến hành dfs để chọn người để trả, ta có 2 cách:

1. Trả cho người ấy với 1 trong số cách có thể trả được.
2. Từ chối trả tiền cho người này.

Ta sẽ thực hiện cho đến khi i=n hoặc số lượng tiền không đủ để đáp ứng nữa sau đó tổng hợp kết quả và so sánh để tìm ra câu trả lời tốt nhất.

# CHƯƠNG TRÌNH VÀ KẾT QUẢ

## Tổ chức chương trình

**4.1.1. Khởi tạo thư viện**

#include <iostream>

#include <fstream>

using namespace std;

**4.1.2. Khởi tạo các cấu trúc dữ liệu và tạo hàm hỗ trơ**

|  |
| --- |
| namespace PAIR{  template<typename T,typename U>  struct pair{  T first;  U second;  pair(){}  pair(T first,U second):first(first),second(second){}  };  }  template <typename Number>  Number \_\_gcd(Number u, Number v) {  while (v != 0) {  Number r = u % v;  u = v;  v = r;  }  return u;  }  template<typename T>  struct vector{  int n;  int capacity;  T \*a;  vector(){  this->n=0;  this->capacity=100;  a=new T[capacity];  }  vector(int size){  this->n=size;  this->capacity=100;  if(n>capacity) capacity=n;  a=new T[capacity];  }  void expand(int newCapacity){  if(newCapacity<=n){  return;  }  T \*old=a;  a=new T[newCapacity];  for(int i=0;i<n;++i) a[i]=old[i];  delete []old;  capacity=newCapacity;  }  int size() const{  return this->n;  }  T& operator[](const int id){  return a[id];  }  void resize(int size){  this->n=size;  capacity=100;  if(n>capacity) capacity=n;  this->a=new T[capacity];  }  void push\_back(T val){  if (n==capacity) expand(2\*n);  a[n]=val;  ++n;  }  void pop\_back(){  --n;  }  void clear(){  n=0;  }  };  void swap(int& a, int& b)  {  int t = a;  a = b;  b = t;  }  int partition (vector<int> arr, int low, int high)  {  int pivot = arr[high];  int i = (low - 1);  for (int j = low; j <= high - 1; j++)  {  if (arr[j] < pivot)  {  i++;  swap(arr[i], arr[j]);  }  }  swap(arr[i + 1], arr[high]);  return (i + 1);  }  void quickSort(vector<int> arr, int low, int high)  {  if (low < high)  {  int pi = partition(arr, low, high);  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  struct customer{  int id;  vector<int> x;  customer(){}  customer(int id,vector<int> x){  x.resize(n);  this->id=id;  this->x=x;  }  void print(ofstream& fo){  fo<<endl;  fo<<"customer ["<<this->id<<"] : "<<endl;  for(int i=0;i<x.size();++i){  fo<<x[i]<<" ";  }  fo<<endl; // error in here  }  void addVector(vector<int> a){  x.resize(a.size());  for(int i=0;i<a.size();++i) x[i]=a[i];  }  }; |

**4.1.3. Khởi tạo biến nhập/xuất từ file và nhập dữ liệu:**

|  |
| --- |
| ifstream fi("F:/DA/input.txt");  ofstream fo("F:/Da/output2.txt");  ofstream fo2("F:/Da/output3.txt");  if(fi.fail()){  return cout<<"ifstream failed "<<endl,0;  }  if(fo.fail()){  return cout<<"ofstream failed "<<endl,0;  }  if(fo2.fail()){  return cout<<"ofstream2 failed "<<endl,0;  }  sum=0;  fi>>n>>m;  a.resize(n);  for(int i=0;i<n;++i) fi>>a[i];  b.resize(n);  for(int i=0;i<n;++i) fi>>b[i];  k.resize(m);  for(int i=0;i<m;++i) fi>>k[i];  v.resize(m);  for(int i=0;i<m;++i){  v[i]=vector<vector<int>>();  } |

**4.1.4.Xuất ra màn hình nội dung đã in.**

|  |
| --- |
| // print out screen content of data  cout<<"--------------------------------------------------"<<endl;  cout<<"CONTENT OF DATA : "<<endl;  cout<<"n : "<<n<<" m : "<<m<<endl;  for(int i=0;i<n;++i){  cout<<"a["<<i+1<<"] = "<<a[i]<<", b["<<i+1<<"] = "<<b[i]<<endl;  }  cout<<"~~~~~~~~~~~~~~"<<endl;  for(int i=0;i<m;++i){  cout<<"k["<<i+1<<"] = "<<k[i]<<endl;  }  cout<<"--------------------------------------------------"<<endl; |

**4.1.5. Xử lý thông tin đầu vào.**

|  |
| --- |
| int g=0;  for(int i=0;i<n;++i) g=\_\_gcd(g,a[i]);  for(int i=0;i<m;++i) g=\_\_gcd(g,k[i]);  for(int i=0;i<n;++i) a[i]/=g;  for(int i=0;i<m;++i) k[i]/=g;  for(int i=0;i<m;++i){  putVec(v[i],k[i]);  } |

**4.1.6 Dùng dfs để tìm số lượng khách tối đa có thể tra**

|  |
| --- |
| vector<customer> mainList;  vector<customer> tmp;  vector<int> cur(n);  for(int i=0;i<n;++i) cur[i]=0;  PAIR::pair<int,vector<int>> ans;  ans.first=0;  get\_ans(0,ans,cur,0,mainList,tmp);  cout<<"ans : "<<ans.first<<endl;  for(int i=0;i<ans.second.size();++i) cout<<ans.second[i]<<" ";  cout<<endl<<"mainList"<<endl;  for(int i=0;i<(int)mainList.size();++i){  for(int j=0;j<mainList[i].x.size();++j){  cout<<mainList[i].x[j]<<" ";  }  cout<<endl;  }  // output in file output2  fo<<"/////////////////////////////////////"<<endl;  fo<<"CUSTOMER LIST"<<endl;  // fo<<"num : "<<(int)mainList.size()<<endl;  for(int i=0;i<(int)mainList.size();++i){  mainList[i].print(fo);  }  fo<<"/////////////////////////////////////"<<endl;  fo<<endl;  fo.close(); |

**4.1.7 Dùng dfs để tìm ra số lượng tiền tối đa có thể tra.**

|  |
| --- |
| maxSum=0;  // output3  quickSort(k,0,(int)k.size()-1);  for(int i=0;i<n;++i) sum+=a[i]\*b[i];  vector<int> tmp2(n);  for(int i=0;i<tmp2.size();++i) tmp2[i]=0;  vector<customer> mainList2;  vector<customer> temp;  dfs(0,tmp2,mainList2,temp);  maxSum\*=g;  fo2<<"max sum : "<<maxSum<<endl;  cout<<"max sum : "<<maxSum<<endl;  for(int i=0;i<mainList2.size();++i){  customer x=mainList2[i];  fo2<<"khach thu ["<<x.id+1<<"] : ";  for(int j=0;j<x.x.size();++j) fo2<<x.x[j]<<" ";  fo2<<endl;  }  fo2.close();  fi.close(); |

**4.1.8 Các hàm hỗ trợ**

|  |
| --- |
| ////////////////////////////////////////////////////////////////////////// QuickSort  int partition (vector<int> arr, int low, int high)  {  int pivot = arr[high];  int i = (low - 1);    for (int j = low; j <= high - 1; j++)  {  if (arr[j] < pivot)  {  i++;  swap(arr[i], arr[j]);  }  }  swap(arr[i + 1], arr[high]);  return (i + 1);  }  void quickSort(vector<int> arr, int low, int high)  {  if (low < high)  {  int pi = partition(arr, low, high);  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  ////////////////////////////////////////////////////////////////////////// Tính tổng trong vector  int sumVec(vector<int>& vec){  int sum=0;  for(int i=0;i<n;++i) sum+=a[i]\*vec[i];  return sum;  }  ////////////////////////////////////////////////////////////////////////// Cộng 2 vector  vector<int> add\_vec(vector<int>& a,vector<int>& b){  int n=a.size();  vector<int> c(n);  for(int i=0;i<n;++i) c[i]=a[i];  for(int i=0;i<n;++i){  c[i]+=b[i];  }  return c;  }  ////////////////////////////////////////////////////////////////////////// Kiểm tra vector hợp lệ  bool good(vector<int> cur){  for(int i=0;i<n;++i){  if(cur[i]>b[i]) return 0;  }  return 1;  }  //////////////////////////////////////////////// Hàm tạo ra các vector hợp lệ cho mỗi người khách  void solve(int id,vector<vector<int>>& vec,int sum,int val,vector<int> cur){  if(sum==val){  vector<int> tmp(cur.size());  for(int i=0;i<cur.size();++i) tmp[i]=cur[i];  vec.push\_back(tmp);  return;  }  if(id==n||sum>val) return;  for(int times=0;times<=b[id];++times){  int add=times\*a[id];  cur[id]=times;  solve(id+1,vec,sum+add,val,cur);  }  cur[id]=0;  }  void putVec(vector<vector<int>>& vec,int val){  vector<int> cur(n);  for(int i=0;i<n;++i) cur[i]=0;  cout<<endl;  solve(0,vec,0,val,cur);  }  //////////////////////////////////////////////// So sánh 2 vector  bool vecGreater(vector<int>& a,vector<int>& b){  int n=a.size();  for(int i=0;i<n;++i){  if(a[i]<b[i]) return 0;  }  return 1;  }  //////////////////////////////////////////////// dfs tìm số lượng người nhiều nhất  void get\_ans(int id,PAIR::pair<int,vector<int>>& ans,vector<int> cur,int cnt,vector<customer>& list,vector<customer> tmp){  if(!good(cur)) return;  if(ans.first<cnt){  ans.first=cnt;  ans.second.resize(cur.size());  for(int i=0;i<cur.size();++i) ans.second[i]=cur[i];  list.resize(tmp.size());  for(int i=0;i<tmp.size();++i){  customer xx=tmp[i];  list[i]=xx;  }  }  if(id==m) return;  // dont choose him  get\_ans(id+1,ans,cur,cnt,list,tmp);  //choose him  for(int i=0;i<v[id].size();++i){  vector<int> tt(v[id][i].size());  for(int j=0;j<v[id][i].size();++j) tt[j]=v[id][i][j];  customer x(id+1,tt);  x.addVector(tt);  tmp.push\_back(x);  vector<int> tttt=add\_vec(cur,v[id][i]);  get\_ans(id+1,ans,tttt,cnt+1,list,tmp);  tmp.pop\_back();  }  }  //////////////////////////////////////////////// dfs tìm số lượng tiền nhiều nhất  void dfs(int id,vector<int> vec,vector<customer>& list,vector<customer> tmp){  if(id==m){  if(!good(vec)) return;  int sumNow=sumVec(vec);  if(maxSum<sumNow){  maxSum=sumNow;  list.resize(tmp.size());  for(int i=0;i<tmp.size();++i){  customer xx=tmp[i];  list[i]=xx;  }  }  return;  }  if(!good(vec)) return;  int sumNow=sumVec(vec);  if(maxSum<sumNow){  maxSum=sumNow;  list.resize(tmp.size());  for(int i=0;i<tmp.size();++i){  customer xx=tmp[i];  list[i]=xx;  }  }  dfs(id+1,vec,list,tmp);  for(int i=0;i<v[id].size();++i){  vector<int> tt(v[id][i].size());  for(int j=0;j<v[id][i].size();++j) tt[j]=v[id][i][j];  customer x(id,tt);  for(int i=0;i<tt.size();++i) x.x[i]=tt[i];  tmp.push\_back(x);  vector<int> tttt=add\_vec(vec,v[id][i]);  dfs(id+1,tttt,list,tmp);  tmp.pop\_back();  }  } |

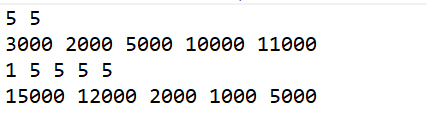
## Ngôn ngữ cài đặt

***Ngôn ngữ lập trình được dùng trong bài toán này là ngôn ngữ lập trình C++ (C plus plus)***

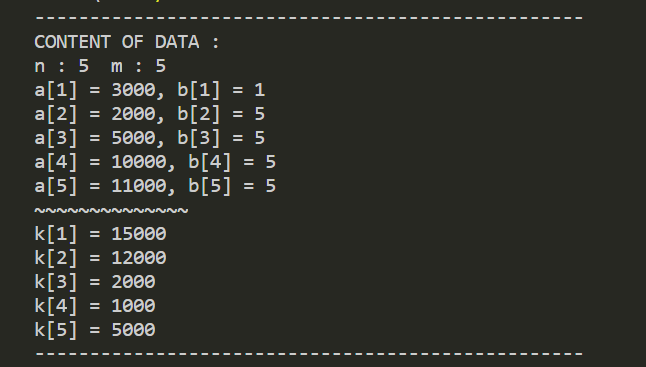
* C++ là một ngôn ngữ lập trinh bậc trung. Nó có nghĩa là bạn có thể sử dụng C++ để phát triển những ứng dụng bậc cao, và cả những chương trình bậc thấp hoạt động tốt trên phần cứng.
* C++ là một ngôn ngữ lập trình hướng đối tượng. Khác với ngôn ngữ lập trình C - một ngôn ngữ lập trình hướng thủ tục, chương trình được tổ chức theo thuật ngữ “chức năng”, một chức năng gồm có những hành động mà bạn muốn làm. C++ được thiết kế với một cách tiếp cận hoàn toàn mới được gọi là lập trình hướng đối tượng, nơi mà chúng ta sử dụng những đối tượng, các lớp và sử dụng các khái niệm như: thừa kế, đa hình, tính đóng gói, tính trừu tượng … Những khái niệm này khá phức tạp, nên nếu bạn chưa hiểu về chúng, đừng lo lắng, chúng ta sẽ lần lượt làm rõ từng khái niệm trong mỗi bài học khác nhau.
* C++ là một ngôn ngữ lập trình hướng cấu trúc giống ngôn ngữ C, nó có nghĩa là chúng ta có thể tổ chức chương trình trên khái niệm functions.
* C++ có thể chạy trên nhiều nền tảng khác nhau như Windows, Mac OS, một số biến thể của UNIX…

## Kết quả

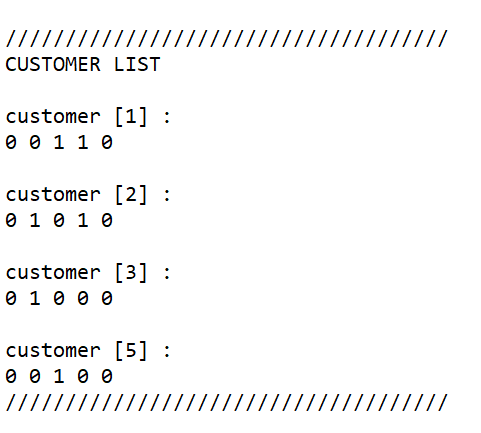
Input :



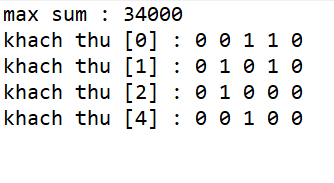
OutPut (console):



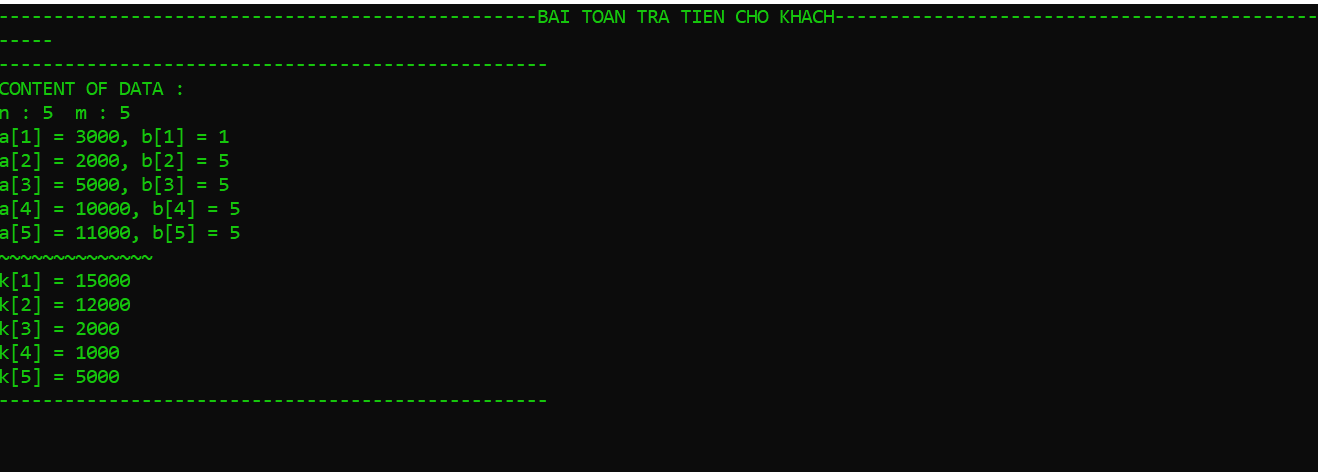
Ouput (output2.txt)



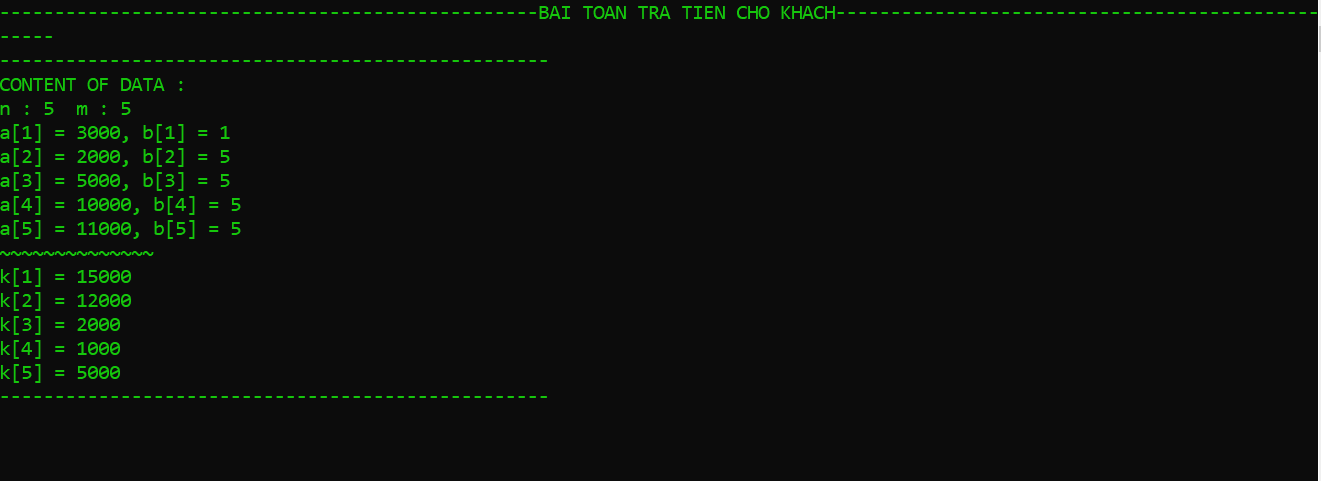
Output (output3.txt)



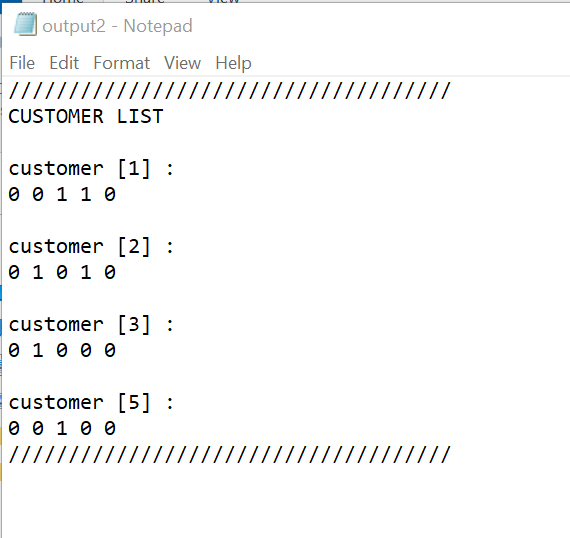
### Giao diện chính của chương trình



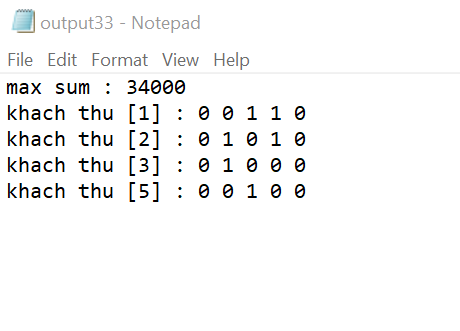
### Kết quả thực thi của chương trình



*Hiển thị ra màn hình nội dung của đầu vào*



*Hiển thị ra số lượng khách nhiều nhất có thể trả và cách trả tiền .*



*Hiển thị ra số lượng tiền lớn nhất có thể trả và cách trả .*

### Nhận xét

Đây là một bài toán khá khó về mặt thuật toán và cài đặt. Tốc độ thuật toán có thể không nhanh bởi vì đây là 1 thuật duyệt tất cả trường hợp nên độ phức tạp là rất cao nếu lượng đầu vào cao thì có thể thuật toán sẽ chạy trong thời gian rất lâu (runtime).

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Qua quá trình nghiên cứu và tìm hiểu đề tài thuật toán thì em đã học được them nhiều kiến thức. Bên cạnh đó em đã có thể cải thiện phần nào về khả năng tổ chức vấn đề và làm sạch code hơn (clean code),…

## Hướng phát triển

-Có thể dùng quy hoạch động mà cụ thể ở đây là Map để lưu lại các kết quả đã đi qua từ đó đỡ phải duyệt lại, giúp tối ưu hơn về mặt thời gian (runtime).

-Có thể dùng 1 vài thư viện để cải thiện về mặt hình thức và giao diện (UI).

TÀI LIỆU THAM KHẢO

QuickSort: <https://www.geeksforgeeks.org/quick-sort/>