**KỸ THUẬT LẬP TRÌNH**

ĐỒ ÁN

QUY HOẠC ĐỘNG VÀ SỐ LỚN

Môn học : **Kỹ thuật lập trình**  
Tên : Phạm Minh Chiến

MSSV : 1612052  
GV : Đậu Ngọc Hà Dương

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ Thông tin

Đại học Khoa học Tự nhiên TP HCM

Tháng 06/2017

## Quy hoạch động

1. **Giới thiệu:**

Phương pháp quy hoạch động được nhà toán học Richard Bellman phát minh năm 1953. Quy hoạch động là kỹ thuật giải các bài toán có bản chất đệ quy nhằm tìm ra kết quả tối ưu cho bài toán trên tinh thần chia để trị, tức chia bài toán lớn thành nhiều bài toán nhỏ, giải tất cả các bài toán nhỏ để tim kết quả của bài toán lớn. Ý tưởng của nguyên lý tối ưu Bellman: “Với mỗi quá trình điều khiển tối ưu, đối với trạng thái bắt đầu A0, với trạng thái A trong quá trình đó, phần quá trình kể từ trạng thái A được xem như trạng thái bắt đầu cũng là tối ưu.” Hay nói cách là “nêu một cấu hình là tối ưu thì mọi cấu hình của nó đều tối ưu”.

1. **Một số khái niệm và các bước cài đặt:**
   1. **Các khái niệm**
      * ***Bài toán quy hoạch đông***: là bài toán giải theo phương pháp quy hoạch.
      * ***Công thức truy hồi***: là công thức phối hợp nghiệm của các bài toán con để có nghiệm bài toán lớn.
      * ***Cơ sở quy hoạch*** (cơ sở): là tập các bài toán nhỏ nhất có ngay lời giải.
      * ***Bảng phương án***: là không gian lưu trữ các kết quả của các bài toán con.
   2. **Yêu cầu:**  
      Một bài toán có thể giải bằng quy hoạch động phải đảm bảo các yếu tố sau:
      * Bài toán lớn phải phân rã được thành nhiều bài toán con mà sự kết hợp kết quả của các bài toán con cho ra được kết quả bài toán lớn.
      * Đảm bảo đủ không gian vật lí cho tất cả các bài toán con.
      * Quá trình từ bài toán con cho ra kết quả bài toán lớn phải là hữu hạn các bước.
   3. **Các bước cài đặt một bài toán bằng quy hoạch động:**
      * Phân tích bài toán và lập công thức truy hồi.
      * Giải tất cả bài toán cơ sở lưu vào bảng phương án (mảng 1, 2 chiều).
      * Dùng công thức truy hồi phối hợp những lời giải của bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải bài toán lớn hơn và lưu vào bảng phương án. Lặp lại cho tới khi tìm được lời giải cho bài toán ban đầu.
      * Dựa vào bảng phương án ta truy ngược để tìm nghiệm tối ưu.
2. **Áp dụng vào bài toán cụ thể:**

Bài toán: Cho thuê máy

Trung tâm tính toán hiệu năng nhận được đơn đặt hang của n khách hàng. Khách hàng i muốn sử dụng máy trong khoảng thời gian từ Si đến Fi và trả tiền thuê là Ci. Hãy bố trí lịch thuê máy để tổng số tiền thu được là lớn nhất mà thời gian sử dụng máy của hai khách bất kì được phục vụ không bằng nhau.

INPUT File

* Dòng 1: N (số khách)
* N dòng tiếp tiếp theo mỗi dòng có cấu trúc: Si Fi Ci

OUTPUT File:

* Dòng 1: tổng số tiền (k khách)
* K dòng tiếp theo mỗi dòng: Si Fi Ci
* Phân tích bài toán:

Đầu tiên ta sắp xếp n đơn đặt hàng theo thứ tự tăng dần thời gian Fi. Ta sẽ thêm 2 phần tử A[0] và A[n+1] vào đơn đặt hàng với A[0] = {MIN,MIN,0}, A[n+1] = {MAX,MAX,0}. Có nghĩa là mọi đơn đặt hàng đều phải xếp sau đơn A[0] và trước đơn A[n+1] và dĩ nhiên đơn A[0], A[n+1] được miễn phí(Ci=0). Đặt Sum(i) là tổng số tiền thu được khi cho thuê các đơn hàng từ A[i] trở đi(sau khi đã sắp xếp). Như vậy bài toán sẽ chọn ra dãy các đơn hàng thỏa:

* Ràng buộc: Không giao nhau, tức sau sắp xếp thì để xếp A[i] vào giữa A[k] và A[k+1] thì phải đảm bảo: A[i].s>=A[k].f và A[i].f<=A[k+1].s
* Kết quả bài toán sẽ là max{Sum(i),1<i<n+1} ,(=Sum(0))
* Cơ sở quy hoạch: trường hợp không cho thuê đơn nào thì kết quả Sum(n+1) = 0.
* Công thức truy hồi:
* Để tính được max(Sum(i)) thì ta cần tính mỗi Sum(i) với I chạy từ n->0. Giá trị Sum(i) được tính theo các Sum(i+1)… Sum(n+1) đã biết như sau:
* Việc chọn A[i] để cho thuê là chính là thêm A[i] vào dãy các đơn đã cho thuê bắt đầu từ A[j] (j>i) sao cho:

+ A[i].f <= A[j].s

+Sum(j) = max với i<j<=n+1,

Vậy ta có công thức:

* Tính Bảng phương án và truy vết:
* Với mỗi Sum(i) = Ci+Sum(j) ta đặt truy vết Trace[i] = j để lưu lại rằng next(i) = j (đơn hàng phía sau đơn A[i] là đơn A[j]). Từ đó ta lần ngược lại theo mảng Trace sẽ được kết quả.
* Ví dụ:

|  |  |
| --- | --- |
| **Input file** | **Output file** |
| 5  1 3 10  5 6 5  3 5 10  4 7 30  6 8 10 | 40  1 3 10  4 7 30 |

**Bảng phương án:**

**Sum**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| S[i],F[i] | MIN, MIN | 1, 3 | 3, 5 | 5, 6 | 4, 7 | 6, 8 | MAX, MAX |
| C[i] | 0 | 10 | 10 | 5 | 30 | 10 | 0 |
| Sum[i] | **40** | 40 | 25 | 15 | 30 | 10 | 0 |
| Trace[i] | 1 | 4 | 3 | 5 | 6 | 6 |  |

**Trace**

Từ bảng phương án ta có 2 lựa chọn:

+ Lựa chọn 1: (1,3) - (4,7) 🡪 tổng tiền: 40

+ Lựa chọn 2: (3, 5) - (5, 6) - (6, 8) 🡪 tổng tiền: 25

* Chọn lựa chọn 1

**\*Code bài cho thuê máy:**

|  |
| --- |
| #include <stdio.h>  #include <conio.h>  #include <stdlib.h>  struct order  {  int s;  int f;  int c;  };  void readData(char\*path, order\*&a, int&n);  void sortData(order\*&a, int n);  void buildSolutionTable(order\*&a, int\*&sum, int\*&trace, int n);  void printTrace(order\*a, int\*sum, int\*trace, int n, char\*path);  void printArray(order\*a, int n, char\*s);  void int()  {  int n;  order \*a;  int \*sum, \*trace;  readData("Thue\_May.txt", a, n);  sortData(a, n);  printArray(a, n, "\nData:");  buildSolutionTable(a, sum, trace, n);  printTrace(a, sum, trace, n,"Thue\_May\_Out.txt");  \_getch();  return 0;  }  void readData(char\*path, order\*&a, int&n)  {  errno\_t err; FILE\*f;  printf("\nReading file: %s", path);  err = fopen\_s(&f, path, "r");  if (err != 0)  {  printf("\nCannot read data!!");  \_getch();  exit(0);  }  else  {  fscanf\_s(f, "%d", &n);  a = new order[n];  int i = 0;  while (!feof(f))  {  int tmp = 0;  fscanf\_s(f, "%d", &tmp);  a[i].s = tmp;  fscanf\_s(f, "%d", &tmp);  a[i].f = tmp;  fscanf\_s(f, "%d", &tmp);  a[i].c = tmp;  i++;  }  fclose(f);  }  }  void buildSolutionTable(order\*&a, int\*&sum, int\*&trace, int n)  {  sum = new int[n + 2];  trace = new int[n + 2];  a = (order\*)realloc(a, (n + 2)\*sizeof(order));  for (int i = n; i >= 1; i--)  a[i] = a[i - 1];  a[0] = { INT\_MIN, INT\_MIN,0};  a[n + 1] = { INT\_MAX, INT\_MAX,0 };  sum[n + 1] = 0;  for (int i = n; i >= 0; i--)  {  int jmax = n + 1;  for (int j = i + 1; j <= n + 1; j++)  {  if (a[i].f <= a[j].s && sum[j] > sum[jmax])  jmax = j;  }  sum[i] = sum[jmax] + a[i].c;  trace[i] = jmax;  }  }  void printTrace(order\*a, int\*sum, int\*trace, int n,char\*path)  {  errno\_t err; FILE\*f;  err = fopen\_s(&f, path, "wb");  if (err != 0)  {  printf("\nCannot write data!!");  \_getch();  exit(0);  }  fprintf(f, "Sum: %d", sum[0]);  int i = trace[0];  while (i != n + 1)  {  fprintf(f,"\n%d %d %d",a[i].s, a[i].f,a[i].c);  i = trace[i];  }  fclose(f);  printf("\nWriting done: %s", path);  }  void sortData(order\*&a, int n)  {  //selection sort  for (int i = 0; i < n; i++)  for (int j = i + 1; j < n - 1; j++)  {  int min = j;  for (int k = j + 2; k < n; k++)  if (a[k].f < a[min].f)  min = k;  if (a[i].f > a[min].f)  {  order tmp = a[i];  a[i] = a[min]; a[min] = tmp;  }  }  }  void printArray(order\*a, int n, char\*s)  {  printf("%s%d\n", s, n);  for (int i = 0; i < n; i++)  printf("(%d,%d) ", a[i].s, a[i].f);  } |

Bài toán Balo dạng 1:Có n món đồ, vật thứ I có trọng lượng là wi và giá trị vi. Hãy chọn ra các món có thể bỏ vào một cái balo có trọng lượng tối đa là W sao cho tổng giá trị các món đồ là lớn nhất.

\* Đầu vào: Tập tin BALO.INP

- Dong 1: Chứa n W

- n dòng tiếp theo, dòng I chưa wi và ai

\* Đầu ra: Tập tin BALO.OUT

- Dong 1: Giá trị lớn nhất trong balo

- Dòng 2: Chỉ số những đồ vật được lấy

**\*Cách giải:**

Gọi ***F(i,j)*** là giá trị lớn nhất có thể có bằng cách chọn món đồ thứ {1, 2, 3, …, i} với giới hạn trọng lượng ***j***. Như vậy **F(n,W)** chính là giá trị lớn nhất khi chọn trong số n gói với giới hạn trọng lượng W.

1. **Công thức truy hồi:**

Với giới hạn trọng lượng ***j***, việc chọn phương án tối ưu trong số các gói {1, 2, 3, …,i-1, i} để có giá trị lớn nhất sẽ xảy ra 2 trường hợp:

* + *Không* chọn gói i: F(i ,j) = F(i-1, j)
  + *Chọn* gói i (wi <= j): F(i ,j) = vi + F(i-1,j - wi)
* Do đó muốn F(i,j) lớn nhất thì F(i,j) phải là số lớn nhất trong 2 trường hợp trên.

1. **Cơ sở quy hoạch đông:**

F(0,j) = 0 với mọi j vì giá trị lớn nhất bang 0 khi chọ trong số 0 món đồ.

1. **Bảng phương án:**

Bảng phương án F gồm n+1 dòng, W+1 cột. Đầu tiên cơ sở quy hoạch động tương đương dòng 0 gồm toàn số 0. Sử dụng công thức truy hồi dùng dòng i-1 để tính dòng i (i<=n)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F** | **0** | **1** | **2** | **…** | **W** |
| **0** | 0 | 0 | 0 | 0 | 0 |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **…** | … | … | … | … | … |
| **n** |  |  |  |  | **x** |

1. **Truy vết:**

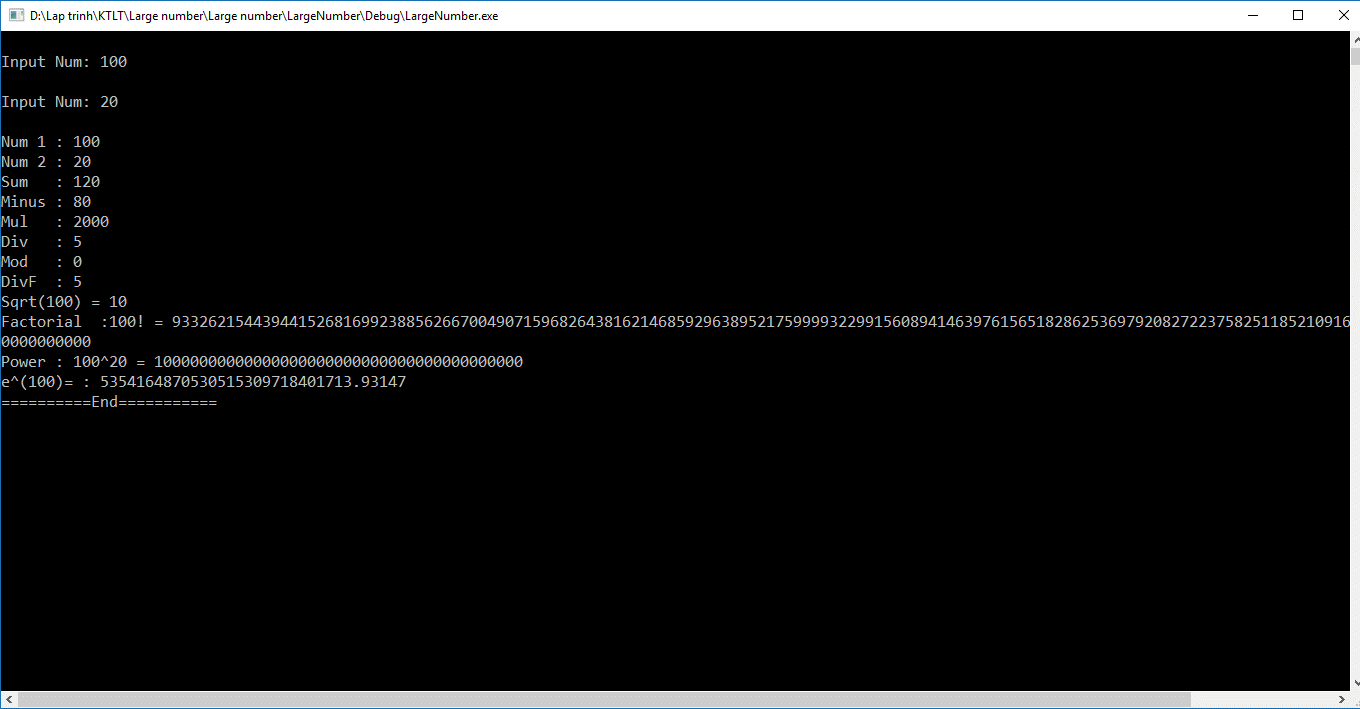
Sau khi tính xong bảng phương án ta sẽ qua tâm đến F(n,W) = x vì đó chính là giá trị lớn nhất thư được trong cả n gói với giới hạn W.   
- Nếu F[n,W] = F[n-1,W] thì không chọn vật thứ n => Truy tiếp F[n-1,W]  
- Nếu F[n,W] != F[n-1,W] thì phép chọn có vật thứ n => Truy tiếp F[n-1,W-wn]  
Cứ tiếp tục cho tới khi truy tới dòng 0.

\*Code bài toán balo dạng 1:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <conio.h>  struct item  {  int w;  int v;  };  void readData(char\*path, item\*&a, int&n, int&W);  void buildSolutionTable(int n, int W, item\*a, int\*\*&F);  void printTrace(int n, int W, item\*a, int\*\*F);  int main()  {  item\*a = NULL;  int n, W,\*\*F;  readData("SUM\_INPUT.txt", a, n, W);  buildSolutionTable(n, W, a, F);  printTrace(n, W, a, F);  \_getch();  return 0;  }  void readData(char\*path, item\*&a,int&n,int&W)  {  errno\_t err;  FILE\*f;  err = fopen\_s(&f, path, "r");  if (err != 0)  {  printf("\nCannot read data"); \_getch(); exit(0);  }  fscanf\_s(f, "%d%d", &n, &W);  a = new item[n];  int i = 1;//==> đồng nhất với bảng phương án  while (!feof(f))  {  int tmp = 0;  fscanf\_s(f, "%d", &tmp);  a[i].w = tmp;  fscanf\_s(f, "%d", &tmp);  a[i].v = tmp;  i++;  }  fclose(f);  }  void buildSolutionTable(int n, int W, item\*a, int\*\*&F)  {  F = new int\*[n + 1];  for (int i = 0; i <= n; i++)  F[i] = new int[W + 1];  for (int j = 0; j <= W; j++)  F[0][j] = 0;//co so quy hoach    for (int i = 1; i <= n; i++)  {  for (int j = 0; j <= W; j++)  {  F[i][j] = F[i - 1][j];//gia su khong cho ai  //neu chon ai tot hon thi ga lai  int tmp = F[i - 1][j - a[i].w] + a[i].v;//giá trị sau khi chọn gói i  if (a[i].w <= j && F[i][j] < tmp)  {  F[i][j] = tmp;//chon ai  }  }  }  }  void printTrace(int n, int W, item\*a, int\*\*F)  {  printf("Weight: %d", W);  while (n != 0)  {  if (F[n][W] != F[n - 1][W])  {  printf("\na[%d]", n);  W -= a[n].w;  }  n--;  }  } |

## Số lớn

1. Giới thiệu:  
   Số lớn là biểu diễn số các số có độ dài mà các kiểu dữ liệu cơ sở không đủ để chứa. Việc lữu trữ các số lớn này phải đảm bảo về mặt bộ nhớ cũng như đúng đắn trong các phép toán.



1. Các phương pháp xử lí số lớn  
   Các cấu trúc dữ liệu dùng để lưu trữ số lớn:  
   - Chuỗi, mảng  
   - Danh sách liên kết

Công việc chính của xử lí số lớn là lưu trữ và tính toán. Sử dụng một chuỗi hoặc mảng đảm bảo cho việc tính toán nhưng không đảm bảo về mặt bộ nhớ vật lí vì chuỗi, mảng thường phải chiếm một vùng nhớ liên tiếp, mà đối với số lớn thì việc này là không thể chấp nhận được. Đó đó giải pháp thay thế là dùng danh sách liên kết.

Ta thiết kế danh sách liên kết có mỗi node chứa data là một kí tự char (1byte) để tiết kiệm tối đa bộ nhớ.

|  |
| --- |
| struct node  {  char data;  node\*pNext;  };  struct stack  {  node\*pHead;  node\*pTail;  }; |

1. Các phép toán trên số lớn
   1. Phép toán logic (>, <, >=, <=, ==, !=):  
      Ta tính độ chênh lệch của 2 số. Kết quả của các phép toán tử là kết quả của phép só sánh độ chên lệch của hai số với 0.

|  |
| --- |
| //operator logical  int compare(stack num1, stack num2)  {  deleteFirstZero(num1); deleteLastZero(num1);  deleteFirstZero(num2); deleteLastZero(num2);  stack st1, st2; initStack(st1); initStack(st2);  int dem = 0, dem1 = 0, dem2 = 0;  //for double  while (!isEmptyStack(num1))  {  if (lastValueStack(num1) != '.')  {  topStack(st1, popEndStack(num1)); dem++;  }  else  {  dem1 = dem;  popEndStack(num1);  }  }  dem = 0;  while (!isEmptyStack(num2))  {  if (lastValueStack(num2) != '.')  {  topStack(st2, popEndStack(num2)); dem++;  }  else  {  dem2 = dem;  popEndStack(num2);  }  }//    dem = dem1 - dem2;  if (dem>0)//chèn vào st2  for (int i = 0; i < dem; i++)  topEndStack(st2, '0');  else  for (int i = 0; i < -dem; i++)  topEndStack(st1, '0');    deleteFirstZero(st1);  deleteFirstZero(st2);  int key = lengthOfStack(st1) - lengthOfStack(st2);  if (key == 0)  {  node\*k1 = st1.pHead;  node\*k2 = st2.pHead;  while (k1->data == k2->data && k1 != st1.pTail&&k2 != st2.pTail)  {  k1 = k1->pNext;  k2 = k2->pNext;  }  key = k1->data - k2->data;  }  return key;  }  bool operator==(stack num1, stack num2)  {  return compare(num1, num2) == 0;  }  bool operator!=(stack num1, stack num2)  {  return compare(num1, num2) != 0;  }  bool operator>(stack num1, stack num2)  {  return compare(num1, num2) > 0;  }  bool operator>=(stack num1, stack num2)  {  return compare(num1,num2) >= 0;  }  bool operator<(stack num1, stack num2)  {  return compare(num1, num2) < 0;  }  bool operator<=(stack num1, stack num2)  {  return compare(num1, num2) <= 0;  } |

* 1. Cộng:

|  |
| --- |
| stack operator+(stack num1, stack num2)  {  stack st\_0=makeStack\_0();    stack st1, st2;  copyStack(num1, st1);  copyStack(num2, st2);  if (isMinus(st1) && isMinus(st2))  {  delMinus(st1); delMinus(st2);  return st\_0 - (st1 + st2);  }  else if (isMinus(st1))  {  delMinus(st1);  return st2 - st1;  }  else if (isMinus(st2))  {  delMinus(st2);  return st1 - st2;  }  fillZeroForDouble(st1, st2);  stack st; initStack(st);  char carray = 0;//nhớ  bool dot = false;  while (!isEmptyStack(st1) || !isEmptyStack(st2))//empty=>pop=0  {  if (lastValueStack(st1) == '.')  {  popEndStack(st1);  popEndStack(st2);  topStack(st, '.');  dot = true;  continue;  }  char tmp = carray + popEndStack(st1) + popEndStack(st2) - 48;  if (tmp >= 58)  {  tmp = tmp - 10;  carray = 1;  }  else  carray = 0;  topStack(st, tmp);  }  if (carray == 1)  topStack(st, '1');  if (dot == true)//có phần thập phân  deleteLastZero(st);  freeStack(st\_0); freeStack(st1); freeStack(st2);  return st;  } |

* 1. Trừ:

|  |
| --- |
| stack operator-(stack num1, stack num2)  {  stack st1, st2;  stack st\_0 = makeStack\_0();  copyStack(num1, st1);  copyStack(num2, st2);  if (isMinus(st1) && isMinus(st2))  {  delMinus(st1); delMinus(st2);  return st2 - st1;  }  else if (isMinus(st1))  {  delMinus(st1);  return st\_0 - (st1 + st2);  }  else if (isMinus(st2))  {  delMinus(st2);  return st1 + st2;  }  fillZeroForDouble(st1, st2);  stack st; initStack(st);  if (st1 == st2)  {  return st\_0;  }  char carray = 0;//nhớ  //xét số nào lớn hơn  bool key = st1 > st2;  bool dot = false;  //tính  while (!isEmptyStack(st1) || !isEmptyStack(st2  {  if (lastValueStack(st1) == '.')  {  dot = true;  popEndStack(st1);  popEndStack(st2);  topStack(st, '.');  continue;  }  char tmp;  if (key>0)  tmp = popEndStack(st1) - carray - popEndStack(st2);  else  tmp = popEndStack(st2) - carray - popEndStack(st1);  if (tmp < 0)  {  carray = 1;  tmp = tmp + 58;  }  else  {  carray=0;  tmp = tmp + 48;  }  topStack(st, tmp);  }  //xóa số 0 ở đầu  deleteFirstZero(st);  if (!key)  topStack(st, '-');  if (dot = true)//có phần thập phaan  deleteLastZero(st);  freeStack(st1); freeStack(st2); freeStack(st\_0);  return st;  } |

* 1. Nhân

|  |
| --- |
| stack operator\*(stack num1, stack num2)  {  stack temp\_1, temp\_2, st; initStack(st); topStack(st, '0');  copyStack(num1, temp\_1); copyStack(num2, temp\_2);  if (isMinus(temp\_1) && isMinus(temp\_2))  {  delMinus(temp\_1); delMinus(temp\_2);  return temp\_1\*temp\_2;  }  else if (isMinus(temp\_1))  {  delMinus(temp\_1);  st = st - (temp\_1\*temp\_2);  return st;  }  else if (isMinus(temp\_2))  {  delMinus(temp\_2);  st = st - (temp\_1\*temp\_2);  return st;  }  stack st1, st2, mul;  initStack(st1); initStack(st2);  //copy to temp stack và tính vị trí dấu '.'  int dem1 = 0, dem2 = 0, dem = 0, n1 = lengthOfStack(num1), n2 = lengthOfStack(num2);  while (!isEmptyStack(num1))  {  if (lastValueStack(num1) != '.')  {  topStack(st1, popEndStack(num1)); dem++;  }  else  {  dem1 = dem;  popEndStack(num1);  }  }  dem = 0;  while (!isEmptyStack(num2))  {  if (lastValueStack(num2) != '.')  {  topStack(st2, popEndStack(num2)); dem++;  }  else  {  dem2 = dem;  popEndStack(num2);  }  }  ////////////////    char carray = 0;//nhớ  int space=0;//số lần lùi  while (!isEmptyStack(st2))//empty=>pop=0  {  initStack(mul);  node\*k1 = st1.pTail;  char m = popEndStack(st2) - 48;  while (k1 != NULL)  {  char tmp = carray + m\*(k1->data - 48);  if (tmp > 9)  {  carray = tmp / 10;  tmp = tmp % 10;  }  else  carray = 0;  topStack(mul, tmp+48);  k1 = getNodeBefore(st1, k1);  }  if (carray > 0)  topStack(mul, carray + 48);  for (int i = 0; i < space; i++)  topEndStack(mul, '0');  st = st + mul;  carray = 0;  space++;  }  //điền dấu .  initStack(mul);  for (int i = 1; i <= dem1 + dem2; i++)  topStack(mul, popEndStack(st));  topEndStack(st, '.');  int decimal = 1;  while (!isEmptyStack(mul))// && decimal <= Max\_Of\_Decimal)  {  topEndStack(st, popStack(mul)); decimal++;  }  deleteFirstZero(st);  deleteLastZero(st);  freeStack(st1); freeStack(st2); freeStack(temp\_1); freeStack(temp\_2); freeStack(mul);  return st;  } |

* 1. Chia
     1. Chia lấy nguyên:

|  |
| --- |
| stack operator/(stack bichia, stack sochia)  {  stack st1, st2,st=makeStack\_0();  initStack(st1);  initStack(st2);  copyStack(bichia, st1);  copyStack(sochia, st2);  if (isMinus(st1) && isMinus(st2))  {  delMinus(st1); delMinus(st2);  return st1/st2;  }  else if (isMinus(st1))  {  delMinus(st1);  st = st - (st1/st2);  return st;  }  else if (isMinus(st2))  {  delMinus(st2);  st = st - (st1/st2);  return st;  }  stack result, tmp, tmp2, tmp3;  initStack(result);  initStack(tmp);  initStack(tmp2);  initStack(tmp3);  if (st2.pHead == st2.pTail && firstValueStack(st2) == '0' || isDouble(st1) || isDouble(st2))  {  return makeError();  }  else if (st2 >= st1)  {  if (st2 == st1)  topStack(result, '1');  else  topStack(result, '0');  return result;  }  while (!isEmptyStack(st1) || tmp >= st2)  {  while (st2>tmp && !isEmptyStack(st1))  {  topEndStack(result, '0');  topEndStack(tmp, popStack(st1));  }  //popEndStack(result);  if (tmp >= st2)  {  for (int i = 9; i >= 1; i--)  {  initStack(tmp2);  initStack(tmp3);  topStack(tmp3, i + 48);  tmp2 = st2\*tmp3;  if (tmp2 <= tmp)  {  topEndStack(result, i + 48);  break;  }  }  tmp = tmp - tmp2;  if (!isEmptyStack(st1))  {  topEndStack(tmp, popStack(st1));  if (isEmptyStack(st1) && tmp<st2)  topEndStack(result, '0');  }  }  else  {  topEndStack(result, '0');  }  }  while (firstValueStack(result) == '0')  popStack(result);  freeStack(st); freeStack(st1); freeStack(st2); freeStack(tmp); freeStack(tmp2);  freeStack(tmp3);  return result;  } |

* + 1. Chia lấy dư:

|  |
| --- |
| stack operator%(stack bichia, stack sochia)  {  if (isMinus(bichia) || isMinus(sochia) || sochia.pHead == sochia.pTail && firstValueStack(sochia) == '0' || isDouble(bichia) || isDouble(sochia))  {  return makeError();  }  return bichia - (bichia / sochia)\*sochia;  } |

* + 1. Chia hai số nguyên (kết quả là số thực):

|  |
| --- |
| stack div2Int(stack bichia, stack sochia)  {  int carry = 0;  stack result, tmp, tmp2, tmp3; initStack(result); initStack(tmp); initStack(tmp2); initStack(tmp3);    result = bichia / sochia;  bichia = bichia % sochia;  //chia hết  if (firstValueStack(bichia) == '0')  return result;    topEndStack(result, '.');  while (carry < Max\_Of\_Decimal &&firstValueStack(bichia)!='0')  {  carry++;  topEndStack(bichia, '0');  tmp = bichia / sochia;  bichia = bichia%sochia;  if (carry == Max\_Of\_Decimal && firstValueStack(tmp) >= '5'&&firstValueStack(tmp) != '9')  ;// topEndStack(result, popStack(tmp) + 1);  else  topEndStack(result, popStack(tmp));  }    while (firstValueStack(result) == '0')  popStack(result);  if (firstValueStack(result) == '.')  topStack(result, '0');  freeStack(tmp); freeStack(tmp2); freeStack(tmp3);  return result;  } |

* + 1. Chia thực (divFloat)

|  |
| --- |
| stack divFloat2(stack bichia, stack sochia)  {  stack temp\_1, temp\_2, st; initStack(st); topStack(st, '0');  copyStack(bichia, temp\_1); copyStack(sochia, temp\_2);  if (isMinus(temp\_1) && isMinus(temp\_2))  {  delMinus(temp\_1); delMinus(temp\_2);  return divFloat2(temp\_1, temp\_2);  }  else if (isMinus(temp\_1))  {  delMinus(temp\_1);  st = st - divFloat2(temp\_1, temp\_2);  return st;  }  else if (isMinus(temp\_2))  {  delMinus(temp\_2);  st = st - divFloat2(temp\_1, temp\_2);  return st;  }  if (!isDouble(bichia) && !isDouble(sochia))  return div2Int(bichia, sochia);  //phép chia 2 số thực  int dem1 = 0, dem2 = 0, dem = 0, n1 = lengthOfStack(bichia), n2 = lengthOfStack(sochia);  stack st1, st2, result;  initStack(st1); initStack(st2); initStack(result);  while (!isEmptyStack(bichia))  {  if (lastValueStack(bichia) != '.')  {  topStack(st1, popEndStack(bichia)); dem++;  }  else  {  dem1 = dem;  popEndStack(bichia);  }  }  dem = 0;  while (!isEmptyStack(sochia))  {  if (lastValueStack(sochia) != '.')  {  topStack(st2, popEndStack(sochia)); dem++;  }  else  {  dem2 = dem;  popEndStack(sochia);  }  }  for (int i = 0; i < dem2;i++)  topEndStack(st1, '0');  for (int i = 0; i < dem1; i++)  topEndStack(st2, '0');  result = div2Int(st1, st2);  deleteLastZero(result);  deleteFirstZero(result);  freeStack(temp\_1); freeStack(temp\_2);  freeStack(st); freeStack(st1); freeStack(st2);  return result;  } |

* 1. Căn bậc hai (sqrt)  
     Giả sử: ⬄ x = y2 ⬄ x/y=y

Nhận xét: Nếu x/y == y thì y chính là kết quả, ngươc lại ta có kết quả là y’ thỏa mãn: y<=y’<=x/y. Ta không thể tìm ngay chính xác được y’ cho nên phải “dò” trong đoạn [y, x/y] bằng cách lấy trong bình cộng, ta dò cho đến khi nào thỏa:

⬄. Ta có sai số của phép “dò”: .

|  |
| --- |
| stack sqrt(stack st)  {  int n = 0;node\*p = st.pHead  for (p; p->data != '.' && p->pNext!= NULL; p = p->pNext)  {  n++;  }  if (n > 3)  n = 3;  else n = 6;  stack complex = makeStack\_0();  topEndStack(complex, '.');  for (int i = 1; i < n;i++)  topEndStack(complex, '0');    topEndStack(complex, '1');  stack kq = makeStack\_1();  stack st\_2 = makeStack\_1();  st\_2 = st\_2 + st\_2;  stack m = abs((kq\*kq - st));  int loop = 0;  while (m>= complex)  {  loop++;  kq = divFloat2(kq + divFloat2(st,kq), st\_2);  m = abs((kq\*kq - st));  }  freeStack(complex);  freeStack(st\_2);  freeStack(m);  return kq;  } |

* 1. Giai thừa:

|  |
| --- |
|  |

* 1. Lũy thừa:  
     Áp dụng khải triển chuỗi Taylor:

|  |
| --- |
| stack powerFloat(stack x, stack y)  {  stack a, st\_1=makeStack\_1(); initStack(a);  a = x - st\_1;  stack s = makeStack\_1();  stack p = makeStack\_1();  stack l = makeStack\_1();  stack gt = makeStack\_1();  stack k = makeStack\_1();  stack tmp; initStack(tmp); stack tmp2; initStack(tmp2);  for (int i = 1; i <= NaN; i++, k = k + st\_1)  {  p = p\*(y - k + st\_1);  l = l\*a;  gt = gt\*k;  tmp = p\*l;  tmp2 = divFloat2(tmp, gt);  s = s + tmp2;  }  freeStack(a);  freeStack(p);  freeStack(l);  freeStack(k);  freeStack(gt);  freeStack(tmp);  freeStack(tmp2);  return s;  } |

* 1. Tính ex

Áp dụng khai triển Taylor:

|  |
| --- |
| stack exp(stack x)  {  stack st\_1 = makeStack\_1();  stack kq = makeStack\_1();  stack mu = makeStack\_1();  stack gt = makeStack\_1();  stack k = makeStack\_1();    for (int i = 1; i <= 3\*NaN; i++, k = k + st\_1)  {  mu = mu\*x;  gt = gt\*k;  kq = kq + divFloat2(mu, gt);  }  freeStack(st\_1);  freeStack(mu);  freeStack(gt);  freeStack(k);  return kq;  } |

* 1. Tính ln(x):

Áp dụng khai triển Taylor:

|  |
| --- |
| stack ln(stack st)  {  if (st <= makeStack\_0())  return makeError();  stack st\_0 = makeStack\_0();  stack st\_1 = makeStack\_1();  stack kq = makeStack\_0();  stack mu = makeStack\_1();  stack gt = makeStack\_1();  stack k = makeStack\_1();  stack x = st - st\_1;  for (int i = 1; i <= 5\*NaN || kq <st\_0; i++, k = k + st\_1)  {  mu = mu\*x;  gt = gt\*k;  if (i%2!=0)  kq = kq + divFloat2(mu, gt);  else  kq = kq - divFloat2(mu, gt);  }  freeStack(st\_1); freeStack(mu); freeStack(gt); freeStack(k); freeStack(x);  return kq;  } |

1. Đánh giá:

Mặc dù là xử lí số lớn nhưng bài làm trên cơ bản là làm đúng trong trường hợp số nhỏ, các trường hợp số lớn thường chạy rất chậm do các thao tác cộng, trừ nhân, chia đều xử lí bằng kiểu char nên qua nhiều bước lặp, kiểm tra, nên chậm hơn so với data kiểu int 4 bytes