підготував старший викладач каф. ЕОМ Козак Назар Богданович

Домашнє завдання №8

Скласти програму (C/C++), яка з застосуванням алгоритмічної стратегії «***динамічне програмування***» дозволяє виконати наступне завдання. Існує T номіналів монет. Видати здачу M мінімальною кількістю монет.

Для порівняння повторно виконати завдання за допомогою ***повного перебору***.

Вибір варіанту

(Nж + Nг + 1) % 6 + 1

де: Nж – порядковий номер студента в групі, а Nг – номер групи(1,2,3,4,5,6,7,8 або 9)

Варіанти завдань

Варіант	N_i		T	M
1	N_0	1	3	11
	N_1	3		
	N_2	6		
2	N_0	1		
	N_1	3		
	N_2	7		
3	N_0	1		
	N_1	3		
	N_2	8		
4	N_0	1		
	N_1	3		
	N_2	9		
5	N_0	1		
	N_1	3		
	N_2	10		
6	N_0	1		
	N_1	3		
	N_2	11		

Приклад коду

Наведений зразок коду демонструє виконання завдання для таких даних:

N_i		T	M
N_0	1	3	11
N_1	4		
N_2	8		

Приклад коду одночасно демонструє виконання домашнього завдання №7 та №8. Вибір демонстрації виконання домашнього завдання №8 здійснюється за допомогою наступного макросу:

```
//#define TASK TASK_A
#define TASK TASK_B
```

Лістинг

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define NANOSECONDS_PER_SECOND_NUMBER 1000000000

//#define TASK TASK_A
#define TASK TASK_B

#define TASK_A 0x00000001
// N0*x0 + N1*x1 + N2*x2 <= M
// TODO: max(K0*x0 + K1*x1 + K2*x2)

#define TASK_B 0x00000002
// N0*x0 + N1*x1 + N2*x2 == M
// TODO: min(x0 + x1 + x2)

#define GLOBAL_CONTEXT_SIZE (13 * sizeof(unsigned int))
#define LOCAL_CONTEXT_1_SIZE (T * sizeof(unsigned int))
#define LOCAL_CONTEXT_2_SIZE ((M + 1) * T * 2 * sizeof(unsigned int))

#define REPEAT_COUNT 10000
#define REPEATOR_PREPARE unsigned int indexIteration;
#define REPEATOR(count, code) \
for (indexIteration = (count); indexIteration--;){ code; }

float getCurrentTime(){
  clock_t time = clock();
  if (time != (clock_t)-1) {
    return ((float)time / (float)CLOCKS_PER_SEC);
  }
```

```
  return 0.; // else
}

#define MAX(a, b) ((a) > (b) ? (a) : (b))

#define T 3
#define M 11

#define N0 1
#define N1 4
#define N2 8

#define K0 2
#define K1 3
#define K2 5

typedef unsigned int Rule_DPS(void * const uArr, unsigned int subsetM, unsigned
int index, void * context, unsigned int level, unsigned int * betterIndex,
unsigned int * valueRule);

unsigned int rule1_DPS(void * const uArr, unsigned int subsetM, unsigned int
index, void * context, unsigned int level, unsigned int * betterIndex, unsigned
int * valueRule){
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])((unsigned int *
const)uArr + 2);

  unsigned int(*const valueF)[T][2] = (unsigned int(*)[T][2])context;

  valueF[subsetM][level][1] += index * uNK0NK1NK2[level][1];

  if (*valueRule > valueF[subsetM][level][1]){
    valueF[subsetM][level][0] = *betterIndex;
    valueF[subsetM][level][1] = *valueRule;
  }
  else{
    *betterIndex = valueF[subsetM][level][0];
    *valueRule = valueF[subsetM][level][1];
  }

  return 0;
}

unsigned int rule2_DPS(void * const uArr, unsigned int subsetM, unsigned int
index, void * context, unsigned int level, unsigned int * betterIndex, unsigned
int * valueRule){
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])((unsigned int *
const)uArr + 2);

  unsigned int(*const valueF)[T][2] = (unsigned int(*)[T][2])context;

  unsigned int asd = subsetM - valueF[subsetM][level][0];
```

```
  if (!index){
    *valueRule = ~0;
  }

  if (valueF[subsetM][level][1] == ~0 || (!level && valueF[subsetM][level][0] !=
subsetM) || *valueRule < (valueF[subsetM][level][1] + index)) {
    valueF[subsetM][level][0] = *betterIndex;
    valueF[subsetM][level][1] = *valueRule;
  }
  else {
    valueF[subsetM][level][1] += index/* * uNK0NK1NK2[level][1]*/;

    *betterIndex = valueF[subsetM][level][0];
    *valueRule = valueF[subsetM][level][1];
  }

  return 0;
}

int run_by_dynamic_programming_strategy(void * const uArr, void * context,
unsigned int level, Rule_DPS rule_DPS){
  unsigned int * const uVC = (unsigned int * const)uArr;
  unsigned int * const uM = (unsigned int * const)uArr + 1;
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])((unsigned int *
const)uArr + 2);
  unsigned int * const uResult = (unsigned int * const)uArr + 2 * *uVC + 2;
  unsigned int * const uScore = (unsigned int * const)uArr + 3 * *uVC + 2;

  unsigned int(*const valueF)[T][2] = (unsigned int(*)[T][2])context;

  unsigned int subsetM, index/*opt*/;

  unsigned int prevBetterIndex = 0, prevMax1 = 0;

  unsigned int vK;

  unsigned int nextSubset = *uM;

  if (level) {
    run_by_dynamic_programming_strategy(uArr, context, level - 1, rule_DPS);
  }

  for (subsetM = 0; subsetM <= *uM; ++subsetM){
    valueF[subsetM][level][0] = 0;
    valueF[subsetM][level][1] = ~0; // index;
    for (index = 0, prevBetterIndex = 0, prevMax1 = 0; (vK = index *
uNK0NK1NK2[level][0]) <= subsetM; ++index){
      valueF[subsetM][level][0] = vK; // index;
      if (level){
        valueF[subsetM][level][1] = valueF[subsetM - vK][level - 1][1];
```

```
      }
      else{
        valueF[subsetM][level][1] = 0;
      }
      if (rule_DPS(uArr, subsetM, index, context, level, &prevBetterIndex,
&prevMax1)){
        break;
      }
    }
  }

  if (level + 1 == *uVC){
    *uScore = valueF[nextSubset][*uVC - 1][1];
    !~*uScore ? ++ * uScore : 0; // no result(~0) fix
    for (index = T; index --> 0;){
      nextSubset -= uResult[index] = valueF[nextSubset][index][0];
      uResult[index] /= uNK0NK1NK2[index][0];
    }
  }

  return 0;
}

void search_rule1(void * const uArr, void * context){
  unsigned int * const uVC = (unsigned int * const)uArr;
  unsigned int * const uM = (unsigned int * const)uArr + 1;
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])((unsigned int *
const)uArr + 2);
  unsigned int * const uResult = (unsigned int * const)uArr + 2 * *uVC + 2;
  unsigned int * const uScore = (unsigned int * const)uArr + 3 * *uVC + 2;

  unsigned int * const contextValues = (unsigned int * const)context;

  unsigned int index;

  unsigned int score = 0;
  unsigned int approximation = 0;

  for (index = 0; index < *uVC; ++index){
    score += contextValues[index] * uNK0NK1NK2[index][1];
  }
  for (index = 0; index < *uVC; ++index){
    approximation += contextValues[index] * uNK0NK1NK2[index][0];
  }

  if (approximation/**valueX0 * *uN0 + *valueX1 * *uN1 + *valueX2 * *uN2*/ <= *uM
&& *uScore < score){
    *uScore = score;
    for (index = 0; index < *uVC; ++index){
      uResult[index] = contextValues[index];
    }
```

```
  }
}

void search_rule2(void * const uArr, void * context){
  unsigned int * const uVC = (unsigned int * const)uArr;
  unsigned int * const uM = (unsigned int * const)uArr + 1;
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])((unsigned int *
const)uArr + 2);
  unsigned int * const uResult = (unsigned int * const)uArr + 2 * *uVC + 2;
  unsigned int * const uScore = (unsigned int * const)uArr + 3 * *uVC + 2;

  unsigned int * const contextValues = (unsigned int * const)context;

  unsigned int index;

  unsigned int score = 0;// *valueX0 + *valueX1 + *valueX2;
  unsigned int approximation = 0;
  for (index = 0; index < *uVC; ++index){
    score += contextValues[index];
  }
  for (index = 0; index < *uVC; ++index){
    approximation += contextValues[index] * uNK0NK1NK2[index][0];
  }
  if (approximation == *uM && (!*uScore || *uScore > score)){
    *uScore = score;
    for (index = 0; index < *uVC; ++index){
      uResult[index] = contextValues[index];
    }
  }
}

void run_search(void * const uArr, void * context, void (search_rule)(void *
const uArr, void * context)){
  unsigned int * const uVC = (unsigned int * const)uArr;
  unsigned int * const uM = (unsigned int * const)uArr + 1;
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])((unsigned int *
const)uArr + 2);
  unsigned int * const uResult = (unsigned int * const)uArr + 2 * *uVC + 2;
  unsigned int * const uScore = (unsigned int * const)uArr + 3 * *uVC + 2;

  unsigned int * const contextValues = (unsigned int * const)context;
  unsigned int * const maxApproximation = &contextValues[3];

  unsigned char index, cf, uMP = *uM + 1;

  unsigned int approximation;
  *uScore = 0;
  *maxApproximation = 0;

  memset(context, 0, *uVC * sizeof(unsigned int));
```

```
  for (cf = 0;;){
    approximation = 0;
    for (index = 0, cf = 1; index < *uVC; ++index){
      //cf ? (++contextValues[index] %= uMP) ? cf = 0 : 0 : 0; // only for C++
      cf ? ++contextValues[index], (contextValues[index] %= uMP) ? cf = 0 : 0 :
0; // for C and C++
      approximation += contextValues[index] * uNK0NK1NK2[index][0];
    }
    if (cf){
      break;
    }
    if (approximation <= *uM){
      search_rule(uArr, context);
    }
  }
}

int compareFunction(const void* a, const void* b){
  const unsigned int(*arg1)[2] = (const unsigned int(*)[2])a;
  const unsigned int(*arg2)[2] = (const unsigned int(*)[2])b;

  return (int)((long long int)(*arg1)[0] - (long long int)(*arg2)[0]);
}

void printResult(char * const title, void * const uArr, unsigned int runTime){
  unsigned int * const uVC = (unsigned int * const)uArr;
  unsigned int * const uResult = (unsigned int * const)uArr + 2 * *uVC + 2;
  unsigned int * const uScore = (unsigned int * const)uArr + 3 * *uVC + 2;

  unsigned int index = 0;

  printf("\r\n%s: ", title);
  printf("\r\nValues: ");
  if (*uScore) {
    for (index = 0; index < *uVC; ++index) {
      printf("X%d=%d ", index, uResult[index]);
    }
    printf("\r\n Score: %d \r\n", *uScore);
  }
  else {
    for (index = 0; index < *uVC; ++index) {
      printf("X%d=? ", index);
    }
    printf("\r\n No solution found \r\n");
  }

  printf("run time: %dns\r\n\r\n", runTime);
}

int main() {
  unsigned int * const uArr = (unsigned int *)malloc(GLOBAL_CONTEXT_SIZE);
```

```
  unsigned int * const context = (unsigned int *)malloc(MAX(LOCAL_CONTEXT_1_SIZE,
LOCAL_CONTEXT_2_SIZE));

  unsigned int * const uVC = uArr;
  unsigned int * const uM = uArr + 1;
  unsigned int(*const uNK0NK1NK2)[2] = (unsigned int(*)[2])(uArr + 2);
  //unsigned int * const uResult = uArr + 2 * *uVC + 2;
  //unsigned int * const uScore = uArr + 3 * *uVC + 2;

  unsigned int(*const valueF)[T][2] = (unsigned int(*)[T][2])context;

  unsigned int level = 0;

  float startTime, endTime;

  REPEATOR_PREPARE;

  if (!(uArr && context)) {
    return 1;
  }

#ifdef PREPARE_DATA
  qsort((void*)uNK0NK1NK2, T, sizeof(unsigned int[2]), compareFunction);
#endif

#if (TASK == TASK_A)
  *uVC = T;
  *uM = M;

  uNK0NK1NK2[0][0] = N0;
  uNK0NK1NK2[1][0] = N1;
  uNK0NK1NK2[2][0] = N2;

  uNK0NK1NK2[0][1] = K0;
  uNK0NK1NK2[1][1] = K1;
  uNK0NK1NK2[2][1] = K2;
#elif (TASK == TASK_B)
  *uVC = T;
  *uM = M;

  uNK0NK1NK2[0][0] = N0;
  uNK0NK1NK2[1][0] = N1;
  uNK0NK1NK2[2][0] = N2;
#endif

  // compute by dynamic programming strategy
  startTime = getCurrentTime();
#if (TASK == TASK_A)
  REPEATOR(REPEAT_COUNT,
    run_by_dynamic_programming_strategy(uArr, context, *uVC - 1, rule1_DPS);
```

```
  );
#elif (TASK == TASK_B)
  REPEATOR(REPEAT_COUNT,
    run_by_dynamic_programming_strategy(uArr, context, *uVC - 1, rule2_DPS);
  );
#endif
  endTime = getCurrentTime();
  printResult((char*)"run by dynamic programming strategy",
    uArr,
    (unsigned int)((endTime - startTime) * (NANOSECONDS_PER_SECOND_NUMBER /
REPEAT_COUNT)));

  // full search
  startTime = getCurrentTime();
#if (TASK == TASK_A)
  REPEATOR(REPEAT_COUNT,
    run_search(uArr, context, search_rule1);
  );
#elif (TASK == TASK_B)
  REPEATOR(REPEAT_COUNT,
    run_search(uArr, context, search_rule2);
  );
#endif
  endTime = getCurrentTime();
  printResult((char*)"search",
    uArr,
    (unsigned int)((endTime - startTime) * (NANOSECONDS_PER_SECOND_NUMBER /
REPEAT_COUNT)));

#if defined(__linux__) || defined(__unix__) || defined(__APPLE__)
  (void)getchar();
#elif defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
  system("pause");
#else
#endif

  return 0;
}
```