1. **Recursive Structure.** Consider an optimum path P from $(0,0)$ to (m,n), and let e be the *last* edge on P. Without loss of generality, suppose e connects $(m-1,n)$ to (m,n)—the argument is essentially the same if e connects $(m,n-1)$ to (m,n). Then, the part of P between $(0,0)$ and $(m-1,n)$ must be an optimum path: if there were a path P' from $(0,0)$ to $(m-1,n)$ with a higher probability of having no accident, then $P'+e$ would be a path to (m,n) with better probability of having no accident than P (a contradiction).

   Table Definition. Define $A[i,j]$ to be the maximum probability of having no accident when riding from $(0,0)$ to (i,j), for $0 \leqslant i \leqslant m$ and $0 \leqslant j \leqslant n$.

   Recurrence Relation. Based on the reasoning above:

$$\begin{aligned}
&A[0,0] = 1 \\
&A[i,0] = A[i-1,0] \times (1-p_{i,0}) && \text{for } 1 \leqslant i \leqslant m \\
&A[0,j] = A[0,j-1] \times (1-q_{0,j}) && \text{for } 1 \leqslant j \leqslant n \\
&A[i,j] = \max\{A[i-1,j] \times (1-p_{i,j}),\ A[i,j-1] \times (1-q_{i,j})\} && \text{for } 1 \leqslant i \leqslant m \text{ and } 1 \leqslant j \leqslant n
\end{aligned}$$

   Bottom-up Algorithm. *(Pseudo-code)*

```
A[0,0] ← 1:
for i ← 1,...,m:
    A[i,0] ← A[i-1,0] × (1 - p_{i,0})
for j ← 1,...,n:
    A[0,j] ← A[0,j-1] × (1 - q_{0,j})
    for i ← 1,...,m:
        A[i,j] ← max{A[i-1,j] × (1 - p_{i,j}), A[i,j-1] × (1 - q_{i,j})}
```

   Solution Reconstruction. Because each value depends on only two others, we can reconstruct an optimum path without the need for an additional array (though it would be acceptable to use one as well). Starting at (m,n), the algorithm works backwards and selects the best alternative (as given by the recurrence) at each step.

```
# Pre-condition: All values A[i,j] have been computed.
(i,j) ← (m,n)
P ← [(i,j)]
while j > 0 or i > 0:
    if j > 0 and A[i,j-1] × (1 - q_{i,j}) ⩾ A[i-1,j] × (1 - p_{i,j]):   j ← j - 1
    else:                                                               i ← i - 1
    P ← [(i,j)] + P
# Post-condition: P contains a path with maximum probability of having no accident.
```

   Total running time is $\Theta(mn)$: $\Theta(mn)$ for computing the values of A and $\Theta(m+n)$ for reconstructing the path.

2. **Recursive Structure.** Consider an optimum solution: a subset $S \subseteq \{1,2,\ldots,n\}$ indicating which coupons to use, such that

$$\left(\sum_{i\in S} p_i\right) + p_b \max\left(0, B - \sum_{i\in S} b_i\right) + p_c \max\left(0, C - \sum_{i\in S} c_i\right) + p_d \max\left(0, D - \sum_{i\in S} d_i\right)$$

   is as small as possible. Then, either $n \in S$ or $n \notin S$. If $n \in S$, then $S - \{n\}$ is an optimum solution to purchase $\max(0, B-b_n)$ burgers, $\max(0, C-c_n)$ fries, and $\max(0, D-d_n)$ drinks using only

coupons from $N_1, \ldots, N_{n-1}$—because if there were a better way to purchase the rest of the food items, we could use it together with coupon N_n to obtain a better solution overall. If $n \notin S$, then S is an optimum solution to purchase B burgers, C fries, and D drinks using only coupons from $N_1, \ldots, N_{n-1}$.

Table Definition. Define $P[i, b, c, d]$ to be the minimum price to purchase at least b burgers, c fries, and d drinks using only coupons from $N_1, \ldots, N_i$ (in other words, no coupon with a number greater than i is used), for $0 \leqslant i \leqslant N$, $0 \leqslant b \leqslant B$, $0 \leqslant c \leqslant C$, and $0 \leqslant d \leqslant D$.

Recurrence Relation. By the reasoning above:

$$P[0, b, c, d] = bp_b + cp_c + dp_d \quad \text{for } 0 \leqslant b \leqslant B, 0 \leqslant c \leqslant C, 0 \leqslant d \leqslant D$$

$$P[i, b, c, d] = \min(P[i-1, b, c, d], p_i + P[i-1, \max(0, b-b_i), \max(0, c-c_i), \max(0, d-d_i)])$$
$$\text{for } 1 \leqslant i \leqslant n, 1 \leqslant b \leqslant B, 1 \leqslant c \leqslant C, 1 \leqslant d \leqslant D$$

Bottom-up Algorithm. *(Pseudo-code)*

```
for b ← 0, 1, ..., B:
    for c ← 0, 1, ..., C:
        for d ← 0, 1, ..., D:
            P[0, b, c, d] ← b·p_b + c·p_c + d·p_d
for i ← 1, 2, ..., n:
    for b ← 0, 1, ..., B:
        for c ← 0, 1, ..., C:
            for d ← 0, 1, ..., D:
                P[i, b, c, d] ← P[i − 1, b, c, d]
                if p_i + P[i − 1, max(0, b − b_i), max(0, c − c_i), max(0, d − d_i)] < P[i, b, c, d]:
                    P[i, b, c, d] ← p_i + P[i − 1, max(0, b − b_i), max(0, c − c_i), max(0, d − d_i)]
```

Solution Reconstruction. Note: The question asked only for the minimum price so technically, simply returning $P[n, B, C, D]$ provides the answer desired. For completeness, we describe here how to compute the optimum subset of coupons.

```
# Pre-condition: All values P[i, b, c, d] have been computed.
S ← ∅
b ← B
c ← C
d ← D
for i ← n, n − 1, ..., 1:
    if P[i, b, c, d] < P[i − 1, b, c, d]:
        S ← S ∪ {i}
        b ← max(0, b − b_i)
        c ← max(0, c − c_i)
        d ← max(0, d − d_i)
# Post-condition: S is an optimum subset of coupons.
```

Total running time is $\Theta(nBCD)$: $\Theta(nBCD)$ for computing the values of P and $\Theta(n)$ for reconstructing the subset.

Alternative solution.

Table Definition $A[i, b, c, d]$: Denote the minimum price to pay for purchasing *at least* b burgers, c fries, and d drinks with the first i coupons only. Note that this does not consider the unit purchase.

Recurrence Relation

$$A[i,b,c,d] = \min\{A[i-1,b,c,d], A[i-1,\max b - b_i, 0, \max c - c_i, 0, \max d - d_i, 0] + p_i, \\ A[i,b+1,c,d], A[i,b,c+1,d], A[i,b,c,d+1]\}$$

Initially, we have $A[0,0,0,0] = 0$. $A[0,b,c,d] = \infty$ if $b \neq 0$, $c \neq 0$, or $d \neq 0$.

Algorithm *Pseudo-code*

$A[0,0,0,0] \leftarrow 0$
$A[0,b,c,d] \leftarrow \infty$, if $b \neq 0$, $c \neq 0$, or $d \neq 0$
$A[i,b,c,d] \leftarrow \infty$, if $b > B$, $c > C$, or $d > D$
for $i \leftarrow 1,\ldots,n$:
  for $b \leftarrow B,\ldots,0$:
    for $c \leftarrow C,\ldots,0$:
      for $d \leftarrow D,\ldots,0$:
        $A[i,b,c,d] \leftarrow \min\{A[i-1,b,c,d],$
          $A[i-1,\max\{b-b_i,0\},\max\{c-c_i,0\},\max\{d-d_i,0\}] + p_i,$
          $A[i,b+1,c,d], A[i,b,c+1,d], A[i,b,c,d+1]\}$
$(B',C',D') = \arg\min\{A[n,B',C',D'] + (B-B')\times p_b + (C-C')\times p_c + (D-D')\times p_d\}$
We use coupons to buy B' burers, C' fries, and D' drinks
We purchase the remaining with unit purchases

Reconstruct Solution We can reconstruct the set of coupon we use to buy B' burgers, C' fries, and D' drinks with the following recursive algorithm.

$CouponSet(n,B',C',D')$:
  if $n = 0$: **return** $\varnothing$
  else if $A[n,B',C',D'] = A[n-1,B',C',D']$:
    return $CouponSet(n-1,B',C',D')$
  else if $A[n,B',C',D'] = A[n,B'+1,C',D']$:
    return $CouponSet(n,B'+1,C',D')$
  else if $A[n,B',C',D'] = A[n,B',C'+1,D']$:
    return $CouponSet(n,B',C'+1,D')$
  else if $A[n,B',C',D'] = A[n,B',C',D'+1]$:
    return $CouponSet(n,B',C',D'+1)$
  else:
    return $CouponSet(n-1,\max\{B'-b_n,0\},\max\{C'-c_n,0\},\max\{D'-d_n,0\}) \cup \{n\}$

3. **Recursive Structure.** Consider a subset of assigments $S \subseteq \{1,\ldots,n\}$—technically, we write S as a subset of *indices*—with total completion time $\sum_{i\in S} t_i \leqslant T$ and maximum total points obtained $p^* = \sum_{i\in S} p_i$. Then, either $n \in S$ or $n \notin S$. If $n \notin S$, then S is a subset of the first $n-1$ assignments that achieves p^* points in time no more than T. If $n \in S$, then $S - \{n\}$ is a subset of the first $n-1$ assignments that achieves $p^* - p_n$ points in time no more than $T - t_n$—and there is no other subset of $\{1,\ldots,n-1\}$ that requires less time to achieve $p^* - p_n$ points, else we would be able to find a better overall solution.

Table Definition. Define $T[i,p]$ to be the minimum time required to obtain at least p points, when completing some of the first i assignemnts $A_1,\ldots,A_i$, for $0 \leqslant i \leqslant n$ and $0 \leqslant p \leqslant P$, where $P = \sum_{k=1}^{n} p_k$ is the total number of points of all the assignments.

Recurrence Relation. By the reasoning above:

$$
\begin{aligned}
&T[0,0] = 0 \\
&T[0,p] = \infty && \text{for } 1 \leqslant p \leqslant P \\
&T[i,p] = \min\{T[i-1,p],\ T[i-1,\max(0,p-p_i)] + t_i\} && \text{for } 1 \leqslant i \leqslant n \text{ and } 0 \leqslant p \leqslant P
\end{aligned}
$$

Bottom-up Algorithm. *(Pseudo-code)*

$P \leftarrow \sum_{i=1}^{n} p_i$
$T[0,0] \leftarrow 0$
for $p \leftarrow 1, \ldots, P$: $\quad T[0,p] \leftarrow \infty$
for $i \leftarrow 1, \ldots, n$:
    for $p \leftarrow 0, \ldots, P$:
        $T[i,p] \leftarrow T[i-1,p]$
        if $p \geqslant p_i$ **and** $T[i,p] > T[i-1,p-p_i] + t_i$:
            $T[i,p] \leftarrow T[i-1,p-p_i] + t_i$

Solution Reconstruction. We can reconstruct the set of selected assignments with the following algorithm.

Pre-condition: All values $T[i,p]$ have been computed.
First, find the maximum number of points possible within time T.
$p \leftarrow P$
while $T[n,p] > T$: $\quad p \leftarrow p-1$
Next, find the assignments that give p points.
$S \leftarrow \varnothing$
for $i \leftarrow n, n-1, \ldots, 1$:
    if $T[i,p] \neq T[i-1,p]$:
        $S \leftarrow S \cup \{i\}$
        $p \leftarrow p - p_i$
Post-condition: S is an optimum subset of assignments.

Total running time is $\Theta(nP)$: $\Theta(nP)$ for computing the values of T and $\Theta(n+P)$ for reconstructing the subset.